Министерство образования и науки Российской Федерации
Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского
Институт информационных технологий, математики и механики
Кафедра математического обеспечения и суперкомпьютерных технологий

Рабочие материалы студента по общему курсу «Методы программирования» (часть 1)

Учебный план подготовки
Бакалавров физико-математических наук по направлению
«прикладная математика и информатика»

Курс второй
Семестр третий
Лекции 36 часов
Практические занятия 36 часов
Лабораторные работы 36 часов
Зачет

Нижний Новгород, 2015

Рабочие материалы к учебному курсу «Методы программирования» подготовил профессор кафедры Программная инженерия, д.т.н., Гергель В.П.

Рабочие материалы заслушаны и утверждены на заседании кафедры математического обеспечения и суперкомпьютерных технологий.
Протокол №1 от 31.08.2015.

Директор Института ИТММ  Гергель В.П.

Содержание

Программа общего курса "ЭВМ и программирование"

1. Цели и задачи курса и его место в учебном процессе на факультете ВМК

1.1. Цель преподавания курса

Усложнение решаемых человеком научно-технических и управленческих задач ведет к возрастанию сложности математических средств, развиваемых для анализа таких проблем. Основой для эффективного использования усложняющихся математических методов специалистами из проблемных областей является создание проблемно-ориентированных человеко-машинных систем, автоматизирующих процесс построения и анализа сложной математической модели объекта или явления (по описанию, представленному в терминах проблемной области). Проблемный специалист может эффективно пользоваться такой системой, не вдаваясь в вопросы программного воплощения соответствующих математических моделей и методов, как при написании программы на языке высокого уровня можно не знать способов трансляции этой программы в машинные команды. По существу такие системы создают некоторую новую проблемно-ориентированную (*виртуальную*) машину для проблемного специалиста, приспособленную для удобного описания объектов проблемной области и операций над этими объектами.

Цель данного курса состоит в изучении основных путей реализации математических методов моделирования и анализа в виде таких виртуальных машин.

1.2. Задачи изучения курса

Изучение курса включает освоение моделей и методов программного отображения на аппаратуру ЭВМ сложных математических моделей (отображающих объекты некоторой проблемной области и операции над ними), обеспечивающих создание виртуальных машин, в т.ч.

- методы представления математических структур, соответствующих сложным объектам (текстам, чертежам и т.п.), и операций над этими структурами;
- методы распределения ресурсов машины между модифицируемыми в процессе обработки структурами;
- методы фиксации шагов обработки как состояний в некотором фазовом пространстве (в результате чего преобразования могут рассматриваться как некоторые выкладки);
- методы указания структур, их частей и операций с помощью системы (виртуальных) обозначений.

1.3. Дисциплины, освоение которых необходимо при изучении данного курса

Курс опирается на материал одноименного вводного курса "ЭВМ и программирование", изучаемого в 1-2 семестрах и направленного на освоение ЭВМ как инструмента автоматизации исполнения алгоритмов обработки информации (общее представление об ЭВМ, понятие алгоритма, способы описания алгоритмов, программа на языке высокого уровня, пропуск задачи на ЭВМ, отладка).

При изучении курса предполагается знание учебного материала общего курса "Основы ЭВМ", дающего сведения о том, каким образом автоматизируется обработка информации на ЭВМ (универсальность вычислительной машины) и каковы пути расширения возможностей ЭВМ по реализации обработки информации. Студенты должны иметь представление о возможностях, которыми обладает аппаратура машин (архитектура ЭВМ) и ее программное расширение (системное обеспечение, работа в среде операционной системы).

В курсе используются основные понятия математической логики (логические переменные и операции двоичной логики), ряд понятий алгебры (алгебраические операции, циклическая группа), теория графов (орографы и их подграфы), дискретной математики (рекурсивные описания, конечные автоматы), понятия функций и математической структуры.

2. Содержание курса

Введение

Важность предмета. Проблема доказательства правильности программ. Способы снижения сложности программного обеспечения.

Цели и задачи курса. Структура учебного плана. Основная и дополнительная литература.

1. Структура действия и структуры данных

1.1. Структуры данных

1.1.1. Разложение действия на элементарные части (структура действия). Порождение структуры операндов структурой действия.

1.1.2. Рекурсия как средство повышения эффективности программирования и определяемая ею собственная структура операндов (векторы, матрицы и др., примеры структур).

1.1.3. Структура алгоритмов и структура данных. Связь с математическим понятием структуры. Графический образ структуры.

1.1.4. Переменные величины и схемы структур. Значения переменных структур и экземпляры схем. Элементы структуры, имена, значения. Основные и вспомогательные базисные множества и отношения в структуре.

1.2. Структуры хранения.

1.2.1. Структуры хранения, представляющие структуры программ.

1.2.2. Структура машинной памяти. Примеры структур хранения данных. Вектор памяти. Массивы. Адресная арифметика как средство задания отношений в структуре хранения. Структуры хранения, операции над структурами и типы.

1.2.3. Использование объектно-ориентированного программирования для реализации структур данных.

1.2.4. Практическая работа 1: Структура хранения для множеств. Характеристический вектор множества и его представление в виде битовой строки. Поэтапное проектирование структуры хранения битовой строки. Спецификация класса TBitField и реализация методов. Разработка класса TSet для представления множеств с использованием структуры данных типа битовая строка.

1.2.5. Практическая работа 2: Структуры хранения для матриц специального вида. Представление на основе двухиндексных массивов и оценка эффективности использования памяти. Исключение хранения ненулевых элементов. Представление матрицы как набора векторов разной длины. Матрица как вектор векторных элементов (шаблоны).

1.3. Динамические структуры.

1.3.1. Переработка информации как преобразование структур данных. Преобразования, приводящие к рекурсивным отношениям исходных и результирующих структур.

1.3.2. Динамические структуры - класс структур с частичным упорядочением (по включению) структур данных, примеры динамических структур (стеки, очереди, деки).

1.4. Динамические структуры и структуры хранения.

1.4.1. Динамические структуры и распределение памяти; средства поддержания динамической структуры. Выражение отношений программными средствами.

1.4.2. Практическая работа 3: Разработка структуры хранения для динамической структуры типа стек. Поэтапная разработка с использованием наследования классов: Обработка кодов завершения программ (класс TDataCom), управления памятью и спецификации методов (класс TDataRoot), реализация операций стека (класс TStack).

1.4.3. Сравнение структур хранения линейных и динамических структур.

1.4.4. Хранение динамических структур при ограниченной памяти. Степень использования памяти. Управление размещением.

1.4.5. Практическая работа 4: Разработка структуры хранения для динамической структуры типа очереди. Введение циклических структур хранения. Инкапсуляция отношения следования. Реализация класса TQueue через наследования от класса TStack.

1.4.6. Хранение нескольких динамических структур и необходимость перераспределения памяти в процессе обработки информации. Пример: хранение двух стеков.

1.5. Динамическое распределение памяти.

1.5.1. Статическое и динамическое распределение памяти. Управление памятью.

1.5.2. Управление памятью путем перепаковки структур хранения, представляющих отношения адресной арифметикой.

1.5.3. Практическая работа 5: Структура хранения нескольких стеков в общем массиве памяти (начальное распределение памяти; переполнение стека; оценка наличия свободной памяти; гипотеза о росте потребности в памяти; перераспределение свободной памяти; перепаковка памяти).

1.5.4. Роль гипотез о росте структур при разработке систем управления памятью. Пример использования гипотезы о сохранении тенденции роста с момента последней перепаковки. Смешанные гипотезы. Оценка параметров модели в ходе выполнения системы (адаптация). Система управления памятью и математическая модель распределения ресурса.

1.6. Распределение памяти для структур хранения, представляющих основные отношения с помощью адресных указателей.

1.6.1. Представление основных отношений с помощью адресных указателей (сцепление). Задание линейных структур сцеплением (ссылки, кванты памяти; звенья; указатель структуры и признак конца). Линейный список.

1.6.2. Хранение динамических структур с использованием сцепления. Реализация списков с использованием языка высокого уровня. Стек свободной памяти. Исключение операций перепаковки.

1.6.3. Практическая работа 6: Реализация структуры хранения нескольких стеков с использованием списков на языке высокого уровня.

1.6.4. Сравнение непрерывной и списковой структур хранения (эффективность организации динамического распределения памяти, необходимость хранения дополнительной информации, возможность прямого способа доступа к данных).

1.6.5. Реализация списков с использованием динамически распределяемой памяти. Пример реализации структуры хранения. Примеры использования стеков: поразрядная сортировка, преобразование арифметических выражений в польскую форму записи.

1.6.6. Практическая работа 7: Разработка общего представления линейного списка для обеспечения списковой структуры хранения. Организация хранения в списках значений разного типа. Проектирование структуры списка. Операции для работы со списком (информационные методы, методы доступа к значениям в списке). Организация навигации по списку (реализация итератора). Вставка и удаление звеньев в списке. Обеспечение удобного интерфейса (безопасное преобразование типов, передача по значению, использование объектов вместо указателей). Разработка шаблона класса-переходника (proxy) для работы со списком конкретного типа значений.

1.6.7. Общая характеристика стандартной библиотеки шаблонов.

2. Динамические структуры и представление на ЭВМ сложных математических моделей

2.1. **Учебно-практическая задача 2.1**: Система для арифметических действий над полиномами.

2.1.1. Упорядочение мономов по степеням переменных (случай одной переменной). Представление полинома вектором коэффициентов при упорядоченных мономах. Арифметические действия над полиномами как операции над векторами (линейными структурами); рекурсия при выполнении операций.

2.1.2. Умножение полиномов и рост структур. Полином как линейная структура в стеке. Использование системы управления стеками для работы с полиномами. Недостатки

представления полинома вектором коэффициентов (расход памяти и времени на хранение и обработку значительного числа нулевых коэффициентов).

2.2. **Учебно-практическая задача 2.2**: Система для арифметических действий над многочленами от нескольких переменных.

2.2.1. Исключение хранения нулевых коэффициентов. Упорядочение мономов по степеням переменных. Индексы мономов. Многочлен как линейная структура, элементы которой имеют значения, определяемые несколькими величинами.

2.2.2. Представление многочленов стеками и проблема перепаковки памяти. Представление многочленов списками. Проблема нулевых многочленов. Общее представление многочленов циклическими списками (понятие циклического списка). Список многочлена, тождественно равного нулю.

2.2.3. Проектирование состава необходимых программ для обеспечения структуры хранения полиномов. Иерархия классов. Разработка программ циклического списка как производного класса от программ линейного списка. Поэтапная разработка программ.

2.2.4. Алгоритм сложения многочленов, содержащий операции управления памятью, включения элементов в середину списка, исключения элемента из списка.

2.3. **Учебно-практическая задача 2.3**: Редактирование текстов.

2.3.1. Текст как линейная структура, элементами которой являются символы. Представление текста линейным списком. Текст как линейная структура, элементами которой являются слова, значения которых есть линейные структуры (последовательности символов). Выражение связи элемента и значения с помощью адресных указателей. Текст как линейная структура, элементами которой являются строки, значения которых есть линейные структуры (последовательности слов). Текст как иерархия линейных структур, образом которой является структура типа дерева, Представление структуры текста связным списком.

2.3.2. Представление текста связным списком из однотипных звеньев. Атомы. Связный список общего вида. Звено как представитель подсписка связного списка. Операция расчленения списка и объединения списков. Свойства основных операций над списком. Пример использования основных операций (выделение списка фамилий из списка пар фамилия-имя; слияние списков имен и фамилий в список пар фамилия-имя).

2.3.3. Обработка списков (как модель обработки текстов). Обход списка; операция "первый атом". Замена атома списка другим атомом или подсписком. Копирование списка. Управление памятью при работе со связными списками (сборка мусора; необходимость маркировки занятых звеньев). Языки для обработки списков.

2.4. **Учебно-практическая задача 2.4**: Структуры хранения геометрических объектов (случай плоского чертежа, содержащего точки и отрезки прямых линий).

2.4.1. Наличие нескольких основных базисных множеств в структуре (точки, линии и т.п.). Представление разнотипных элементов структуры звеньями одинакового формата (использование сцепления для выражения принадлежности точек линиям). Плексы. Различия чертежа и графа представляющего его плекса.

2.4.2. Алгоритм обхода плекса. Плекс как представление выражения (операторы, операнды, значения). Вычисление выражения, представленного плексом (построение рисунка или чертежа). Плексы как представление арифметических выражений. Общее выражение, представляемое плексом.

Учебный курс "Методы программирования-2", ВМК ННГУ, 2 курс
План проведения практических занятий

Занятие	Тема
1-2	ООП. Наследование и шаблоны. Перегрузка операций. Полиморфизм. Примеры: Комплексная арифметика, строки, вектора, графическая библиотека
3	**Практическая работа 1:** Структура хранения множеств. Завершение реализации класса TBitField - конструктор копирования, - метод ClrBit, - перегрузка операции присваивания, - перегрузка операции стандартного вывода Разработка класса TSet
4	**Практическая работа 2:** Структуры хранения для матриц специального вида. Завершение реализации класса TVector - перегрузка операции сравнения, - перегрузка операции сложения, - перегрузка операции стандартного ввода Завершение реализации класса TMatrix - метод GetValue, - перегрузка операции доступа к элементу вектора, - перегрузка операции сравнения, - перегрузка операции стандартного ввода Разработка структуры хранения матриц как набора векторов разной длина (подход 3)
5-6-7	**Лабораторная работа 1**. Стеки
8-9-10	**Лабораторная работа 2**. Очереди
11-12	**Практическая работа 7:** Разработка общего представления линейного списка для обеспечения списковой структуры хранения. Завершение реализации классов TList. Общая характеристика стандартной библиотеки шаблонов STL
13-15	**Лабораторная работа 3**. Полиномы
16	Зачетное занятие

Нижегородский Государственный Университет им. Н.И. Лобачевского

Общий курс:

Методы программирования - 2

Введение

Гергель В.П., профессор
кафедра МО ЭВМ ВМК

Содержание

- Важность предмета
- Фундаментальные основы применения ЭВМ
- Цели и задачи курса
- Структура учебного плана
- Литература
- Горести и радости ремесла

Важность предмета…

Причины широкого использования ВТ

- Компьютеры – интересная область человеческой деятельности (фантастические темпы развития)
  - За 50 лет скорость вычислений возросла более чем в 1000 млн. раз
  - При таких темпах развития автомобили весили бы 200 г, тратили бы 2 л бензина на 1.5 млн.км, и стоили бы 2.5$
- Компьютеры – искусственный интеллект
  - Шенкс (Англия XIX век) за 20 лет работы вычислил число π с точностью 707 знаков (в 520 знаке ошибка) - на ЭВМ в начале 50 вычислено более 500 тыс. знаков
- Моисеев Н.Н. – в истории человечества можно выделить только три равнозначных по важности события
  - Приручение огня
  - Изобретение парового двигателя
  - Создание компьютера

Важность предмета…

ЭВМ - выход из информационного тупика

- мосты 10^5-10^6,
- аэродинамика 10^9-10^{11},
- гидродинамика 10^{12}-10^{14},
- управление экономикой – 10^{16} (~10 млрд.человек)

Важность предмета…

Думаю,что в мире вряд ли найдётся покупателей больше,чем на 5 вычислительных машин.

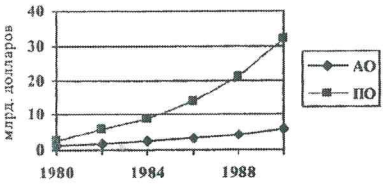
Томас Уотсон, президент IBM, 1943

Нет никаких причин,чтобы люди захотели завести компьютеры у себя дома.

Кен Олсен, президент DEC, 1977

Важность предмета

Соотношение затрат на АО и ПО



Фундаментальные основы применения ЭВМ
(по материалам учебных дисциплин 1 курса)

Основы ЭВМ
- Двоичный принцип представления информации в ЭВМ
- Декомпозиция операторов программы
- Универсальность ЭВМ

Методы программирования
- Структурное программирование
- Модульное программирование
- Концепция типа
- Объектно-ориентированное программирование (ООП)

Цели и задачи курса…

Пример 1. Проверка правильности программы вычисления площади треугольника (Майерс Г.)
1. Правильный неравносторонний
2. Правильный равносторонний
3. Правильный равнобедренный (3 варианта)
4. Одна сторона =0
5. Одна сторона <0
6. Сумма двух сторон равна третьей
7. Сумма двух сторон меньше третьей
8. Длины всех сторон равны нулю
9. Не все исходные значения заданы...

Цели и задачи курса…

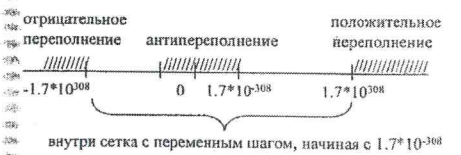
Пример 2. Проверка умножителя (Дейкстра Э.)
- Пусть размер ячейки 27 бит.
- Тогда для полной проверки (исчерпывающее тестирование) имеется 2^{54} вариантов.
- Если время операции умножение несколько десятков мксек, то полный тест потребует около 10 тыс.лет (!)

Цели и задачи курса…

Пример 3. Арифметика вещественных чисел
- $A = m*16^p$, где **m** - мантисса (характеристика), **p** - порядок
  - масса Солнца $2*10^{38}$ г (значащих цифр ~ 5)
  - масса электрона $9*10^{-28}$ г (значащих цифр ~ 9)
- Си **double** 8 байт от $1.7*10^{-308}$ до $1.7*10^{308}$, значащих цифр 15-16 (порядок – 11 бит, мантисса – 52 бит)
- **long double** 10 байт от $3.4*10^{-4932}$ до $1.1*10^{4932}$, значащих цифр 19 (порядок – 15 бит, мантисса – 64 бит)

Цели и задачи курса…

Шкала представимых чисел



Цели и задачи курса…

Рассмотрим влияние такого представления чисел на примере численного вычисление производной

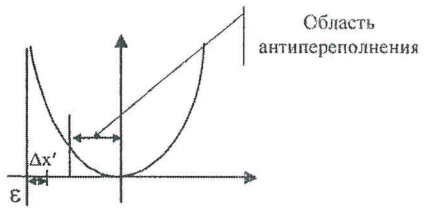
$$\frac{df(x)}{dx} = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x} \approx \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

Пусть $f(x)=x^2$. Возьмём $\varepsilon<0$.

Точное значение $\frac{f(x)}{dx} = 2\varepsilon$

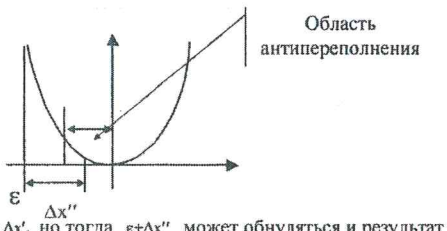
Приближенное значение (1) $\frac{(\varepsilon+\Delta x)^2 - \varepsilon^2}{\Delta x} = (2)\Delta x + \varepsilon + \varepsilon$

Цели и задачи курса…



$\Delta x' << 1$ может попасть в область антипереполнения – деление на ноль, т.е. слишком малые значения нельзя

Цели и задачи курса…



$\Delta x'' > \Delta x'$, но тогда $\varepsilon+\Delta x''$ может обнуляться и результат будет равен $f'(\varepsilon) \approx -\frac{\varepsilon^2}{\Delta x}$

Используем соотношение (2), результат будет равен ε (зависит от порядка выполнения операций)

Цели и задачи курса…

Свойства машинной арифметики с плавающей запятой

1. Представляет конечное множество, но не континуум
2. Не обеспечивает замыкание +, *
3. Может не выполняться ассоциативность
4. Нейтральный элемент не единственен
5. Единичный элемент не единственен, точнее 1 нет, поскольку 0.9999...9 в многократной степени не равно самому себе
6. Сравнение на совпадение не существует

Цели и задачи курса…

Примеры сложного ПО

1.	ПО Шаттл	500 тыс. строк
2.	ПО СОИ	10 млн. (70% от стоимости) т.е. 5 тыс. томов по 300 стр.

Цели и задачи курса…

Основная проблема:

- программа решает поставленную задачу
- программа не содержит ошибок

Методика решения проблемы:

- математически адекватное моделирование (*анализ*) задачи
- *декомпозиция* алгоритмов и данных (разбиение на части) и формализованное (теоретически обоснованное) построение структур действий и данных
- синтез (*агрегация*) программы из элементов (модулей, конструктивных элементов), при обеспечении правильности порождаемого ПО

Цели и задачи курса

- модели и методы (математические основы) разработки сложных программных систем
- математические методы анализа и синтеза программ
- структуры данных и конструирование математических моделей

Структура учебного плана

- Лекции – 72 часа (2 часа в неделю)
- Практикум – 72 часов (2 часа в неделю)
- Лабораторные работы - 72 часов (2 часа в неделю)

Отчетность – зачет (3 семестр)
– экзамен (4 семестр)

☞ Конкурс показательных лабораторных работ (4 семестр)

Литература

Основная литература

1. Топп У., Форд У. Структуры данных в C++.- М. Бином, 1999
2. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ.- МЦМО, 1999
3. Вирт Н. Алгоритмы+структуры данных=программы. - М.: Мир, 1985
4. Страуструп Б. Язык программирования C++. – М.: Бином, 2001.
5. Гергель В.П. и др. Методы программирования. Учебное пособие. Н.Новгород: ННГУ, 1997

Дополнительная литература

1. Хьюз Ч., Пфлигер Ч., Роуз Л. Методы программирования: курс на основе ФОРТРАНА.- М.: Мир, 1981. Гл.5. Структуры данных
2. Мейер Б., Бодуэн К. Методы программирования, т.1,2.- М.: Мир, 1982
3. Лэнгсам Й., Огенстайн М., Тененбаум А. Структуры данных для ПЭВМ.- М.: Мир, 1989 (Бейсик)
4. Любимский Э.З., Мартынюк В.В., Трифонов Н.А. Программирование.- М.: Наука, 1980

Горести и радости ремесла (Брукс)

Радости ремесла

- это абсолютная радость творчества
  - как ребёнок радуется, стряпая пирожки из песка
  - как весьма умный занимался сотворением мира
- это радость создания вещей, полезных другим людям
- это очарование, заключённое в самом процессе создания сложных, загадочных объектов

ЭВМ обладает притягательной силой бильярда или музыкального автомата, доведённого до логической завершенности

- это возможность постоянно учиться, вытекающая из непрерывно меняющегося характера задачи
- это удовольствие работать с очень гибким материалом

Программист, как поэт, работает почти исключительно головой. Очень редко материал для творчества допускает такую гибкость.

Горести и радости ремесла (Брукс)

Горести ремесла

- надо работать очень тщательно как минёр
- задачи, стоящие перед программистом, определяют другие люди
- приятно выдавать большие идеи, но какой же поистине "адский труд" иногда поиск самой крошечной ошибки
- продукт к моменту завершения устарел

Программирование - одновременно и асфальтовая топь, поглощающая многие начинания, и творческая деятельность с присущими только ей радостями и горестями.

Горести и радости ремесла (Ершов А.П.)

Программист должен обладать способностью первоклассного математика к абстракции и логическому мышлению, в сочетании с эдисоновским талантом соорудить всё, что угодно, из нуля и единицы. Он должен сочетать аккуратность бухгалтера с проницательностью разведчика, фантазию автора детективных романов с трезвой практичностью экономиста. А кроме того, программист должен иметь вкус к коллективной работе, понимать интересы пользователя и многое другое.

Заключение

- Декомпозиция и анализ структуры ПО – путь к построению надежного ПО
- Освоение подхода
  - Изучение методов декомпозиции математических моделей при их реализации на ЭВМ
  - Освоение существующего набора структур данных, используемых для построения средств поддержки математических моделей

Вопросы для обсуждения

- Какие существуют способы уменьшения сложности ПО ?
- Существуют ли программы без ошибок ?

Следующая тема

- Структуры данных и структуры действий

Нижегородский Государственный Университет им. Н.И. Лобачевского

Общий курс:

Методы программирования - 2

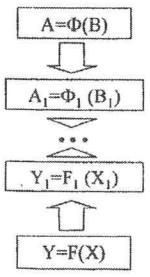
Тема 1:

Структура действия и структуры данных

Гергель В.П., профессор
кафедра МО ЭВМ ВМК

Содержание

Общая схема отображения математических моделей на ЭВМ...



(**A** – исходные данные, **B** – выходные данные, **Φ** – правило преобразования)

При отображении модели на ЭВМ данные модели и необходимые преобразования реализуются при помощи уже существующих объектов на ЭВМ

Таких уровней декомпозиции может быть несколько

Подобная ситуация прослеживается и при разработке программ для автоматизации деятельности ЭВМ (реализация уровней программного обеспечения – снизу вверх)

Общая схема отображения математических моделей на ЭВМ...

📖 Отображение математических моделей на аппаратуру ЭВМ можно себе представить как последовательность этапов построения иерархически-согласованных моделей

Сверху вниз - этапы построения всё более конкретных и детальных моделей, ориентированных на отображение на аппаратуру ЭВМ

Снизу вверх - этапы построения всё более общих моделей, более приближённых к объектам исследования

Общая схема отображения математических моделей на ЭВМ...

Как правило, между моделями верхнего и нижнего уровня остаётся несколько нереализованных промежуточных слоёв и ликвидация этого разрыва и есть *основная задача программиста.*

Общий аппарат для построения программных систем - **Структуры данных и структуры действий**

1.1. Структуры данных

1. Структуры данных, порождаемые структурой действия

ЭВМ является универсальной, поскольку все операторы любого алгоритма разлагаются в последовательность базовых операций.

Разложение оператора рождает разложение операнда

Пример: Скалярное произведение

$$(a,b)= a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

Как располагать значения ?

$$a = (a_1,a_2,a_3) \text{ или } a = (a_3,a_1,a_2)$$

☑ Структура операндов определяется структурой действия

1.1. Структуры данных

2. Структуры данных, для которых возможна рекурсивные вычисления…

Запишем алгоритм скалярного произведения для произвольной длины вектора

$$\sum_0 = 0, \ldots \sum_i = \sum_{i-1} + a_i * b_i$$

Результат шага даётся через результат предшествующего шага. Такое описание называется *рекурсией*. Даёт краткое описание процесса.

Индексы можно убрать в Σ, если используется значение только последней частичной суммы.

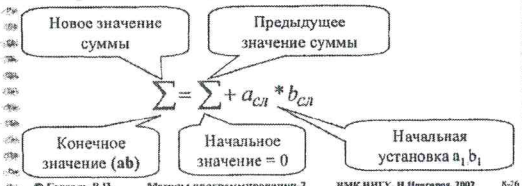
$$\sum = 0, \ldots \sum = \sum + a_i * b_i$$

1.1. Структуры данных

2. Структуры данных, для которых возможна рекурсивные вычисления…

Пусть:

- Элементы векторов располагаются последовательно слева направо,
- Имеется указатель текущего элемента,
- Определена операция "следующий элемент"



1.1. Структуры данных

2. Структуры данных, для которых возможна рекурсивные вычисления

🕮 Введение рекурсии (реализация циклов) требует установления *отношения следования* между элементами данных (т.е. введение понятия соседства и перехода к следующему).

☑ Использование рекурсивно (итеративно) описанного действия предполагает наличие структуры операндов и эта структура является **свойством операндов**.

1.1. Структуры данных

3. Понятие структуры данных…

Одной из наиболее общих математических абстракций является понятие **алгебраической системы**

< A, O, R >

где

A - множество операндов

O - множество операций $A^{n_i} \to A, \; i \in I$

R - множество отношений $r_j \subseteq A^{m_j}, \; j \in J$

n_i - арность операций, m_j - арность отношений

Если операций нет ***модель (структура)***

Если отношений нет ***универсальная алгебра***

1.1. Структуры данных

3. Понятие структуры данных…

Определение 1.1. *Математическая структура*

$S=(M_1,\ldots,M_k; p_1,\ldots,p_s)$

есть одно или несколько множеств $M_1,\ldots,M_k$, элементы которых находятся в некоторых отношениях $p_1,\ldots,p_s$.

$M_1,\ldots,M_k$ – *базисные множества* структуры.

Каждое отношение p_i есть двоичная функция, аргументами которой являются элементы базисного множества. Если аргументы функции находятся в отношении, то значение p_i = ИСТИНА.

Определение 1.2. *Структура данных* есть модель данных в виде математической структуры.

1.1. Структуры данных

3. Понятие структуры данных…

Примеры

- множество операндов и операций и порядок их записи (*арифметическое выражение*),
- множество узлов детали и порядок их соединения (*чертеж*),
- множество людей и родственные связи между ними (*генеалогическое дерево*),
- множество населенных пунктов и пути сообщения (*карта*)

3. Понятие структуры данных…

Пример 1.1. Вектор $a=(a_1,\ldots,a_n)$

$$M_a=\{a_1,\ldots,a_n\}$$

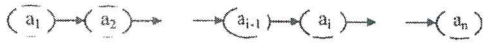
$$p_a\{a_i,a_j\}=\begin{cases}и, & j=i+1\\ л, & j\neq i+1\end{cases}$$

Модель вектора (структура данных)

$S_a=(M_a, p_a)$

3. Понятие структуры данных…

☑ Структуры с *бинарными отношениями* допускают случай графического изображения
- элементы множества изображаются точками или кружками;
- пары (a_i,a_j), для которых отношение истина, соединяются стрелкой от первого аргумента ко второму.



☑ Образ структуры с бинарными отношениями - ***ориентированный граф.***

3. Понятие структуры данных

Определение 1.3. Структуры, которым соответствует ориентированный граф с вершинами, лежащими на одной ломаной, называют *линейными.*

Есть два особых элемента:
- начальный элемент a_1: $(\forall j)\ p(a_j,a_1)=л$ не имеет предшествующего элемента;
- конечный элемент a_n: $(\forall j)\ p(a_n,a_j)=л$ не имеет следующего элемента.

4. Понятие схемы и экземпляра структуры данных…

$(-8, 5, 0) \in R^3$ – вектор с конкретными числовыми значениями

$(a_1, a_2, a_3) \in R^3$ – вектор – переменная

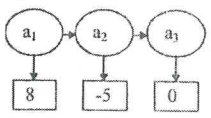
☑ *Переменная* – множество значений и имя, которому можно присвоить конкретное значение.

S_a – схема структуры

S_a^* – экземпляр структуры (экземпляр) в соответствии с КОДАСИЛ

4. Понятие схемы и экземпляра структуры данных …

Экземпляр



Наличие конкретных значений для элементов можно выразить при помощи отношения "*иметь значение*"

$p_1^*(a_i, \alpha_i)=и$, если a_i имеет значение α_i

4. Понятие схемы и экземпляра структуры данных …

Определение 1.4. Структура данных $S_a^*=(M_a, R; p_a, p_1^*)$ с установленными значениями элементов, называется *экземпляром.*

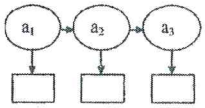
☑ p_1^* - отражает не отношения между элементами, а индивидуальные свойства элемента

☑ Изолированные элементы множества, не связанные стрелками, не изображаются на графе.

1.1. Структуры данных

4. Понятие схемы и экземпляра структуры данных …

Схема структуры



Отношение p_1 является переменной величиной.

Определение 1.5. Структура данных

$$S_a=(M_a, R; p_a, p_1),$$

которая соответствует рассмотрению структуры как переменной величины, называется *схемой структуры.*

1.1. Структуры данных

4. Понятие схемы и экземпляра структуры данных

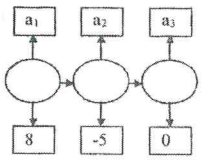
Определение 1.6. Отношения делят на две части:
- отношения, описывающие отношение следования элементов и необходимые для рекурсивно описанных операций, называемые *основными* (по ним и ведется классификация структур);
- прочие отношения описывают индивидуальные свойства элементов, и называются *вспомогательными.*

☑ Алгоритм соответствует схеме структуры. Вычисления соответствует экземпляру.

1.1. Структуры данных

5. Именование элементов структуры…

Различимость элементов данных, необходимая для указания этих элементов, обеспечивается путем присваивания им уникальных имен.



1.1. Структуры данных

5. Именование элементов структуры…

Наличие имен для элементов можно выразить при помощи отношения "*иметь имя*".

N – множество имен

p_2 – отношение "иметь имя"

$S_a^*=(M_a, R, N; p_a, p_1^*, p_2^*)$ – экземпляр

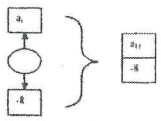
$S_a=(M_a, R, N; p_a, p_1, p_2)$ – схема

1.1. Структуры данных

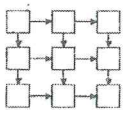
5. Именование элементов структуры

Пример 1.2. Матрица $A=(a_{ij})$

Элемент матрицы



Матрица



☒ Формальное определение матрицы ?

☒ Есть ли начальные и конечные элементы ?

Заключение

- Понятие структуры данных
- Линейные структуры
- Схема и экземпляр структуры
- Базисные и вспомогательные отношения
- Примеры структур данных

Вопросы для обсуждения

- Роль выделения структур данных при разработке программ
- Способы представления структур в ЭВМ

Следующая тема

- Структуры хранения данных

Нижегородский Государственный Университет им. Н.И. Лобачевского

Общий курс:

Методы программирования - 2

Тема 2:

Структуры хранения данных

Гергель В.П., профессор
кафедра МО ЭВМ ВМК

Содержание

1. Структура программы…

Определение 1.7. Машинный образ абстрактной структуры данных называется *структурой хранения данных.*

Структура программы:

1. Структура программы

☑ Для хранения команд используются ячейки памяти

☑ Отношение следования команд реализуются счетчиком команд или командами передачи управления

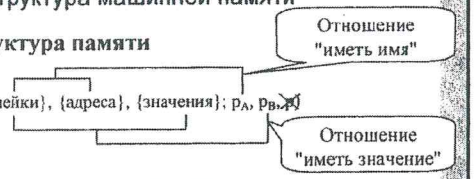
☞ Аппаратура ЭВМ приспособлена для хранения программ

2. Структура машинной памяти …

Структура элемента памяти

2. Структура машинной памяти

Структура памяти



☑ Отношение следования между элементами памяти реализовано программным путем (в отличие от структур программ)

☑ Программа может реализовать только те отношения, которые могут быть охарактеризованы как отношения целых чисел (адресов)

3. Структура хранения вектора …

Возможный способ хранения вектора – использование непрерывной области памяти

α + 0	α + 1	α + 2
-8	5	0

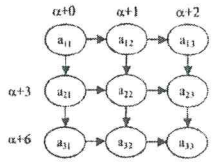
- ☑ Два элемента являются соседними, если адрес одного из них отличается на единицу от адреса другого
- ☑ Такое размещение не фиксируется в аппаратуре, а должно быть реализовано программой
- ☑ Значение α (база) – может играть роль имени всего вектора, смещение до элемента определяется номером элемента

3. Структура хранения вектора …

Определение 1.8. Структура хранения подобного типа (т.е. последовательность однотипных элементов единиц памяти с адресами, возрастающих на единицу), обеспечивающая реализацию абстрактных линейных структур данных (векторов), обычно называется *вектором памяти* или *одноместным (одноиндексным) массивом*.

3. Структура хранения вектора …

Пример 1.3. Матрица $A=(a_{ij})$



Адрес
$(a_{ij}) = \alpha + 3*(i-1) + (j-1)$

Каждый элемент матрицы участвует в двух отношениях (по строке и по столбцу):
- Следующий по строке – адрес + 1
- Следующий по столбцу – адрес + 3

4. Использование ООП для представления структур данных

Для реализации структур данных на ЭВМ наиболее адекватным подходом является *объектно-ориентированное программирование*:
- **класс** – схема структуры данных;
- **объект** – экземпляр структуры;
- **поля объекта** могут использоваться для хранения элементов структуры, а **методы** – для реализации отношений.

Заключение

- Необходимость разработки программ для реализации отношений
- Использование непрерывных участков памяти для хранения элементов структуры
- Понятие вектора памяти
- Примеры структур хранения: вектора и матрицы
- Использование ООП при реализации структур данных на ЭВМ

Вопросы для обсуждения

- Практическое использование изученного теоретического материала

Темы заданий для самостоятельной работы

- Реализация вектора и матрицы с использованием ООП
- Реализация класса TString для символьных строк

Следующая тема

- Примеры разработки структур хранения

Нижегородский Государственный Университет им. Н.И. Лобачевского

Общий курс:

Методы программирования - 2

Практическая работа 1:

Структура хранения множества

Гергель В.П., профессор
кафедра МО ЭВМ ВМК

Содержание

1. Анализ задачи

☑ *Множество* – набор элементов
☑ Для множества определены операции:
- проверка наличия элемента $a \in A$
- добавление элемента $A+a$
- удаление элемента $A-a$

☑ Теоретико-множественные операции
- объединение $A \cup B$
- пересечение $A \cap B$
- вычитание $A \setminus B$

☑ *Универс* U – множество всех элементов

2. Проектирование…

Конкретизация (допущения и ограничения)
- элементы множества проиндексированы (каждому элементу соответствует уникальный индекс)
- множество индексов элементов составляют непрерывный диапазон целых значений

Тогда любое множество $A \subset U$ может быть описано *характеристическим вектором*

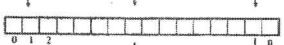
$$\alpha=(\alpha_1\ \alpha_2 \ldots\ \alpha_n),\quad \alpha_i=\begin{cases}\alpha_i=1 \Leftrightarrow\ \alpha_i \in A\\ \alpha_i=0, \text{ иначе}\end{cases}$$

2. Проектирование…

Множество $\alpha=(\alpha_1\ \alpha_2 \ldots\ \alpha_n)$

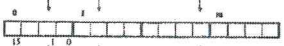
Представление

Битовая строка



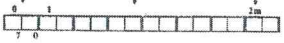
Представление

Массив битовых элементов



Представление

Оперативная память (обратный порядок хранения)



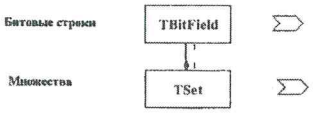
2. Проектирование

☑ Нумерация бит в битовой строке – слева направо
☑ Нумерация элементов в массиве – слева направо, биты элемента – справа налево
☑ Байты двухбайтового элемента располагаются в ОП в обратном порядке (сначала байт с младшими битами, затем байт со старшими битами) – поддержка отображения на аппаратном уровне

↳ При реализации целесообразно выделить базовый класс TBitField, обеспечивающий представление битовых строк
- последовательность разработки
- создание стандартного класса

Множества: структура хранения

3. Реализация



Контрольный пример: программа, приложение

Заключение

- Стадии программной разработки (анализ, проектирование, реализация, доказательство правильности)
- Поэтапная разработка программ (иерархия и наследование)
- Стиль программирования

Вопросы для обсуждения

- Расширение набора операций для множества
- Реализация с использованием шаблонов

Темы заданий для самостоятельной работы

- Завершение разработки класса TBitField
- Реализация класса TSet
- Реализация класса TSet с использованием шаблонов
- Реализация класса TSet через наследование TBitField

Следующая тема

- Разработка структуры хранения для матриц специального типа

```
// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// bitfield.h - Copyright (c) Гергель В.П. 07.05.2001
//
// Битовые поля

#ifndef __BITFIELD_H
#define __BITFIELD_H

#include <iostream.h>
#include <stdlib.h>

typedef unsigned int TELEM;

class TBitField {
  private:
    int  BitLen; // длина битового поля - макс. к-во битов
    TELEM *pMem; // память для представления битового поля
    int  MemLen; // к-во эл-тов Мем для представления бит.поля
    // методы реализации
    int   GetMemIndex ( const int n ) const; // индекс Мем для бита n     (#О2)
    TELEM GetMemMask  ( const int n ) const; // битовая маска для бита n (#О3)
  public:
    TBitField(int len);              // (#О1)
    TBitField(const TBitField &bf); // (#П1)
    ~TBitField();                    // (#С)
    // доступ к битам
    int GetLength ( void ) const;      // получить длину (к-во битов) (#О)
    void SetBit ( const int n );       // установить бит           (#О4)
    void ClrBit ( const int n );       // очистить бит             (#П2)
    int  GetBit ( const int n ) const; // получить значение бита   (#Л1)
    // битовые операции
    int  operator==(const TBitField &bf);       // сравнение       (#О5)
    TBitField & operator=(const TBitField &bf); // присваивание    (#П3)
    TBitField operator|  (const TBitField &bf); // операцмя "или"  (#О6)
    TBitField operator&  (const TBitField &bf); // операцмя "и"    (#Л2)
    TBitField operator~  ( void );              // отрицание       (#С)
    friend istream &operator>>(istream &istr, TBitField &bf);       // (#О7)
    friend ostream &operator<<(ostream &ostr, const TBitField &bf); // (#П4)
};
// Структура хранения битового поля
//   бит.поле - набор битов с номерами от 0 до BitLen
//   массив pMem рассматривается как последовательность MemLen элементов
//   биты в эл-тах pMem нумеруются справа налево (от младших к старшим)
// О8 Л2 П4 С2
#endif

// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// bitfield.cpp - Copyright (c) Гергель В.П. 07.05.2001
//
// Битовые поля

#include "bitfield.h"

TBitField :: TBitField(int len) : BitLen(len) {
  MemLen = ( len + 15 ) >> 4; // в эл-те pMem 16 бит (TELEM==int)
  pMem   = new TELEM[MemLen];
  if ( pMem != NULL )
    for ( int i=0; i<MemLen; i++ ) pMem[i] = 0;
}              /*-------------------------------------------*/
```

```
TBitField :: TBitField(const TBitField &bf) { // конструктор копирования
  BitLen = bf.BitLen;                                          // SKIP_ON
  MemLen = bf.MemLen;
  pMem   = new TELEM[MemLen];
  if ( pMem != NULL )
    for ( int i=0; i<MemLen; i++ ) pMem[i] = bf.pMem[i];       // SKIP_OFF
}             /*-------------------------------------------*/

TBitField :: ~TBitField() {
  delete pMem;                                                 // SKIP_ON
  pMem = NULL;                                                 // SKIP_OFF
}             /*-------------------------------------------*/

int TBitField :: GetMemIndex ( const int n ) const { // индекс Мем для бита n
  // преобразовать к int и разделить на 16
  return n >> 4; // в эл-те pMem 16 бит
}             /*-------------------------------------------*/


TELEM TBitField :: GetMemMask ( const int n ) const { // битовая маска для бита
n
  // преобразовать к int, найти остаток от деления на 16 и сдвинуть
  return 1 << (n & 15);
}             /*-------------------------------------------*/

// доступ к битам битового поля

int TBitField :: GetLength(void) const { // получить длину (к-во битов)
  return BitLen;
}             /*-------------------------------------------*/

void TBitField :: SetBit ( const int n ) { // установить бит
  if ( (n > -1) && (n < BitLen) )
    pMem[GetMemIndex(n)] |= GetMemMask(n);
}             /*-------------------------------------------*/

void TBitField :: ClrBit ( const int n ) { // очистить бит
  if ( (n > -1) && (n < BitLen) )                              // SKIP_ON
    pMem[GetMemIndex(n)] &=~GetMemMask(n);                     // SKIP_OFF
}             /*-------------------------------------------*/

int TBitField :: GetBit ( const int n ) const { // получить значение бита
  if ( (n > -1) && (n < BitLen) )                              // SKIP_ON
    return pMem[GetMemIndex(n)] & GetMemMask(n);
  return 0;                                                    // SKIP_OFF
}             /*-------------------------------------------*/

// битовые операции

TBitField & TBitField :: operator=(const TBitField &bf) { // присваивание
  BitLen = bf.BitLen;                                          // SKIP_ON
  if ( MemLen != bf.MemLen ) {
    MemLen = bf.MemLen;
    if ( pMem != NULL ) delete pMem;
    pMem   = new TELEM[MemLen];
  }
  if ( pMem != NULL )
    for ( int i=0; i<MemLen; i++ ) pMem[i] = bf.pMem[i];
  return *this;                                                // SKIP_OFF
}             /*-------------------------------------------*/
```

```
int  TBitField :: operator==(const TBitField &bf) {  // сравнение
  int res = 1;
  if ( BitLen != bf.BitLen ) res = 0;
  else
    for ( int i=0; i<MemLen; i++ )
      if ( pMem[i] != bf.pMem[i] ) { res = 0; break; }
  return res;
}              /*-------------------------------------------*/


TBitField TBitField :: operator| (const TBitField &bf) {  // операция "или"
  int i, len = BitLen;
  if ( bf.BitLen > len ) len = bf.BitLen;
  TBitField temp(len);
  for ( i=0; i<MemLen;    i++ ) temp.pMem[i]  =    pMem[i];
  for ( i=0; i<bf.MemLen; i++ ) temp.pMem[i] |= bf.pMem[i];
  return temp;
}              /*-------------------------------------------*/


TBitField TBitField :: operator& (const TBitField &bf) {  // операция "и"
  int i, len = BitLen;                                    // SKIP_ON
  if ( bf.BitLen > len ) len = bf.BitLen;
  TBitField temp(len);
  for ( i=0; i<MemLen;    i++ ) temp.pMem[i]  =    pMem[i];
  for ( i=0; i<bf.MemLen; i++ ) temp.pMem[i] &= bf.pMem[i];
  return temp;                                            // SKIP_OFF
}              /*-------------------------------------------*/

TBitField TBitField :: operator~ ( void ) {  // отрицание
  int len = BitLen;                                       // SKIP_ON
  TBitField temp(len);
  for ( int i=0; i<MemLen; i++ ) temp.pMem[i]  =   ~pMem[i];
  return temp;                                            // SKIP_OFF
}              /*-------------------------------------------*/

// ввод/вывод

istream &operator>>(istream &istr, TBitField &bf) { // ввод
  // формат данных - последовательность из 0 и 1 без пробелов
  // начальные пробелы игнорируются
  // при получении не 0 или 1 - завершение ввода
  int i=0; char ch;
  // поиск {
  do { istr >> ch; } while (ch != ' ');
  // ввод элементов и включение в множество
  while (1) { istr >> ch;
    if ( ch == '0' ) bf.ClrBit(i++);
    else if ( ch == '1' ) bf.SetBit(i++); else break;
  }
  return istr;
}              /*-------------------------------------------*/

ostream &operator<<(ostream &ostr, const TBitField &bf) { // вывод
  // формат данных - последовательность 0 и 1                // SKIP_ON
  // вывод элементов
  int len = bf.GetLength();
  for ( int i=0; i<len; i++ )
    if ( bf.GetBit(i) ) ostr << '1'; else ostr << '1';
  return ostr;                                            // SKIP_OFF
}              /*-------------------------------------------*/
```

```
// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// set.h - Copyright (c) Гергель В.П. 07.05.2001
//
// Множества

#ifndef __SET_H
#define __SET_H

#include "bitfield.h"

class TSet {
  private:
    int  MaxPower; // максимальная мощность множества
    TBitField BitField;
  public:
    TSet(int mp);
    TSet(const TSet &s);       // конструктор копирования
    TSet(const TBitField &bf); // конструктор преобразования типа
    operator TBitField();
    // доступ к битам
    int GetMaxPower ( void ) const;      // максимальная мощность множества
    void InsElem  ( const int n )      ; // включить элемент в множество
    void DelElem  ( const int n );       // удалить элемент из множества
    int  IsMember ( const int n ) const; // проверить наличие элемента в мн-е
    // теоретико-множественные операции
    int  operator==(const TSet &s);  // сравнение
    TSet & operator=(const TSet &s); // присваивание
    TSet operator+  (const int n);   // включение элемента в множество
    TSet operator-  (const int n);   // удаление элемента из множества
    TSet operator+  (const TSet &s); // объединение
    TSet operator*  (const TSet &s); // пересечение
    TSet operator~  ( void );        // дополнение
    friend istream &operator>>(istream &istr, TSet &bf);
    friend ostream &operator<<(ostream &ostr, const TSet &bf);
};
#endif

// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// set.cpp - Copyright (c) Гергель В.П. 04.10.2001
//
// Множество - реализация через битовые поля

#include "set.h"

TSet :: TSet(int mp) : MaxPower(mp), BitField(mp) {}

TSet :: TSet(const TSet &s) : // конструктор копирования
  MaxPower(s.MaxPower), BitField(s.BitField) {
}

TSet :: TSet(const TBitField &bf) : // конструктор преобразования типа
  MaxPower(bf.GetLength()), BitField(bf){
}

TSet :: operator TBitField() { // преобразование типа к TBitField
  TBitField temp(this->BitField);
  return temp;
}

// максимальная мощность и проверка принадлежности элементов

int TSet :: GetMaxPower ( void ) const { // получить макс. к-во эл-тов
  return MaxPower;
}              /*-------------------------------------------*/
```

```
int TSet :: IsMember(const int Elem ) const { // элемент мн-ва ?
  return BitField.GetBit(Elem);
}              /*-----------------------------------------*/

// включение/исключение элемента множества

void TSet :: InsElem (const int Elem ) { // включение элемента множества
  BitField.SetBit(Elem);
}              /*-----------------------------------------*/

void TSet :: DelElem ( const int Elem ) { // исключение элемента множества
  BitField.ClrBit(Elem);
}              /*-----------------------------------------*/

// теоретико-множественные операции

TSet & TSet :: operator=(const TSet &s) { // присваивание
  BitField = s.BitField;
  MaxPower = s.GetMaxPower();
  return *this;
}              /*-----------------------------------------*/

int  TSet :: operator==(const TSet &s) {  // сравнение
  return BitField == s.BitField;
}              /*-----------------------------------------*/

TSet TSet :: operator+ (const TSet &s) {  // объединение
  TSet temp(BitField | s.BitField);
  return temp;
}              /*-----------------------------------------*/

TSet TSet :: operator* (const TSet &s) {  // пересечение
  TSet temp(BitField & s.BitField);
  return temp;
}              /*-----------------------------------------*/

TSet TSet :: operator~ ( void ) {  // дополнение
  TSet temp(~BitField);
  return temp;
}              /*-----------------------------------------*/

// перегрузка ввода/вывода

istream &operator>>(istream &istr, TSet &s) { // ввод
  // формат данных - { i1, i2,.., in }
  int temp; char ch;
  // поиск {
  do { istr >> ch; } while (ch != '{');
  do {   // ввод элементов и включение в множество
    istr >> temp; s.InsElem(temp);
    do { istr >> ch; } while ( ( ch != ',' ) && ( ch != '}' ) );
  } while (ch != '}');
  return istr;
}              /*-----------------------------------------*/

ostream &operator<<(ostream &ostr, const TSet &s) { // вывод
  // формат данных - { i1, i2,.., in }
  int i, n; char ch = ' ';
  ostr << "{";
  n = s.GetMaxPower();
  for ( i=0; i<n; i++ ) {   // вывод элементов
    if ( s.IsMember(i) ) { ostr << ch << ' ' << i; ch = ','; }
  }
  ostr << " }";
  return ostr;
}              /*-----------------------------------------*/
```

```
// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// Copyright (c) Гергель В.П. 05.10.2001
//
// Тестирование множества

#pragma hdrstop
#include <iomanip.h>
#include <conio.h>
#include "bitfield.cpp"
#include "set.cpp"

//---------------------------------------------------------------------------

#pragma argsused
int main(int argc, char* argv[]) {
  int n, m, k, count;
  cout << "Тестирование программ поддержки понятия множества" << endl;
  cout << "                Решето Эратосфена" << endl;
  cout << "Введите верхнюю границу целых значений - ";
  cin  >> n;
  TSet s(n+1);
  // заполнение множества
  for ( m=2; m<=n; m++ ) s.InsElem(m);
  // проверка до sqrt(n) и удаление кратных
  for ( m=2; m*m<=n; m++ )
    // если м в s, удаление кратных
    if ( s.IsMember(m) )
      for ( k=2*m; k<=n; k+=m )
       if ( s.IsMember(k) ) s.DelElem(k);
  // оставшиеся в s элементы - простые числа
    cout << "Печать простых чисел" << endl;
  count = 0; k = 1;
  for ( m=2; m<=n; m++ )
    if ( s.IsMember(m) ) {
      count++;
      cout << setw(3) << m << " ";
      if ( k++ % 10 == 0 ) cout << endl;
    }
  cout << endl;
  cout << "В первых " << n << " числах " << count << " простых" << endl;
  getch();
  return 0;
}
//---------------------------------------------------------------------------
```

Нижегородский Государственный Университет им. Н.И. Лобачевского

Общий курс:

Методы программирования - 2

Практическая работа 2:

Структуры хранения для матриц специального вида

Гергель В.П., профессор
кафедра МО ЭВМ ВМК

Содержание

1. Ленточные матрицы

$$\begin{pmatrix} a_{11} & a_{12} & & & \\ a_{21} & a_{22} & a_{23} & & \\ & a_{32} & a_{33} & a_{34} & \\ & & a_{43} & a_{44} & a_{45} \\ & & & a_{54} & a_{55} \end{pmatrix}$$

☑ Для хранения элементов можно выделить непрерывный вектор памяти размера 3*n-2

☑ **Адрес** $(a_{ij}) = \alpha + 3*(i-1) + (j-i)$

2. Треугольные матрицы...

$$\begin{pmatrix} a_{11} & a_{12} & a_{23} & \dots & a_{1n} \\ & a_{22} & a_{23} & \dots & a_{2n} \\ & & a_{33} & \dots & a_{3n} \\ & & & \dots & \\ & & & & a_{nn} \end{pmatrix}$$

Подход 1

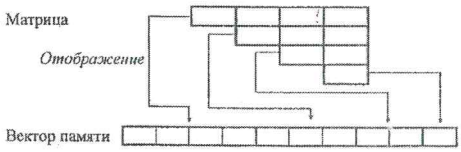
Матрицы подобного вида можно представить как матрицы общего вида и использовать для хранения двухиндексные массивы

- используется память $V_{исп} = n^2$
- необходимая память $V_{необ} = n(n+1)/2$

↳ Эффективность использования памяти $E_{исп} \approx 0.5$

2. Треугольные матрицы...

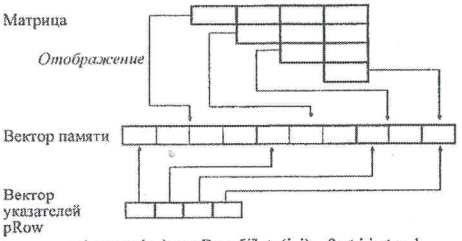
Подход 2 Исключение хранения элементов ниже главной диагонали



Адрес $(a_{ij}) = \alpha + i*n - i*(i-1)/2 + (j-i),\ 0 \le i,j \le n-1$

2. Треугольные матрицы...

Подход 2 Исключение элементов ниже главной диагонали (ускорение доступа)



Адрес $(a_{ij}) = pRow[i] + (j-i),\ 0 \le i,j \le n-1$

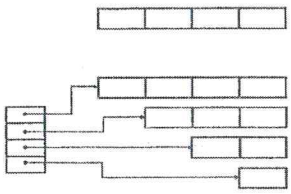
Структуры хранения для матриц специального типа

2. Треугольные матрицы...

Подход 3 Представление матрицы в виде набора векторов

Вектор

Матрица



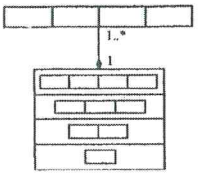
Структуры хранения для матриц специального типа

2. Треугольные матрицы...

Подход 4 Матрица как вектор векторных элементов (шаблоны)

Вектор

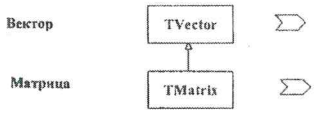
Матрица



Структуры хранения для матриц специального типа

2. Треугольные матрицы...

Подход 4 Матрица как вектор векторных элементов (шаблоны)



Контрольный пример: программа, приложение

Заключение

- Многовариантный анализ возможных способов разработки ПО
- Оценка эффектности использования памяти
- Использование шаблонов для автоматической генерации однотипного кода

Вопросы для обсуждения

- Реализация разных вариантов представления треугольных матриц (подходы 2 и 3)
- Достоинства и недостатки применения шаблонов

Темы заданий для самостоятельной работы

- Завершение реализации классов TVector и TMatrix
- Разработка структуры хранения матриц как набора векторов разной длина (подход 3)

```
// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// utmatr4.h - Copyright (c) Гергель В.П. 07.05.2001
//
// Верхние треугольные матрицы - реализация на основе шаблона вектора

#ifndef __TMATRIX_H
#define __TMATRIX_H

#include <conio.h>
#include <iomanip.h>
#include <iostream.h>

template <class ValType>
  class TVector {
    protected:
      ValType *pVector;
      int Size;       // размер вектора
      int StartIndex; // индекс первого элемента вектора
    public:
      TVector(int s=10, int si=0 );               //                          (#О1)
      TVector(const TVector &v);                  // конструктор копирования  (#Л1)
     ~TVector();                                  //                          (#О2)
      int GetSize()        { return Size;       } // размер вектора           (#О)
      int GetStartIndex()  { return StartIndex; } // индекс первого элемента  (#О)
      ValType & GetValue   (int pos);            // доступ с контролем индекса(#П1)
      ValType & operator[] (int pos);            // доступ                   (#П2)
      int       operator==(const TVector &v);    // сравнение                (#П3)
      TVector  & operator= (const TVector &v);    // присванивание            (#О3)
      // скалярные операции
      TVector   operator+ (const ValType &val);  // прибавить скаляр         (#Л2)
      TVector   operator- (const ValType &val);  // вычесть скаляр           (#С1)
      TVector   operator* (const ValType &val);  // умножить на скаляр       (#С2)
      // векторные операции
      TVector   operator+ (const TVector &v);    // сложение                 (#С3)
      TVector   operator- (const TVector &v);    // вычитание                (#С4)
      TVector   operator* (const TVector &v);    // скалярное произведение   (#С5)
      // ввод-вывод
      friend istream & operator>>( istream &in, TVector &v) {              //(#П4)
        for ( int i=0; i < v.Size; i++ ) in >> v.pVector[i];         // SKIP_ON
        return in;                                                   // SKIP_OFF
      }
      friend ostream & operator<<( ostream &out, const TVector &v) {       //(#С6)
        for ( int i=0; i < v.Size; i++ ) out << v.pVector[i] << ' '; // SKIP_ON
        return out;                                                  // SKIP_OFF
      }
};

template <class ValType>
TVector<ValType>::TVector (int s, int si ) {
  pVector = new ValType[s];
  Size = s; StartIndex = si;
}

template <class ValType>
TVector<ValType>::TVector(const TVector<ValType> &v) { //конструктор копирования
  pVector = new ValType[v.Size];                                  // SKIP_ON
  Size = v.Size; StartIndex = v.StartIndex;
  for ( int i=0; i < Size; i++ ) pVector[i] = v. pVector[i]; //  SKIP_OFF
}
template <class ValType>
TVector<ValType>::~TVector () {
  delete[] pVector;
}
```

```
template <class ValType>
ValType & TVector<ValType>::operator[] (int pos) { // доступ без конроля индекса
  return pVector[pos-StartIndex];                              // SKIP_ON SKIP_OFF
}

template <class ValType>
TVector<ValType> & TVector<ValType>::operator=(const TVector &v){ // присваивание
  if ( this != &v ) {
    if ( Size != v.Size ) {
      delete[] pVector;
      pVector = new ValType[v.Size];
    }
    Size = v.Size; StartIndex = v.StartIndex;
    for ( int i=0; i < Size; i++ ) pVector[i] = v.pVector[i];
  }
  return *this;
}

template <class ValType>
TVector<ValType> TVector<ValType>::operator+(const TVector<ValType> &v) { //
сложение
  TVector temp(Size,StartIndex);                               // SKIP_ON
  for ( int i=0; i < Size; i++ )
    temp.pVector[i] = pVector[i] + v.pVector[i];
  return temp;                                                 //  SKIP_OFF
}

// Верхние треугольные матрицы

template <class ValType>
  class TMatrix : public TVector<TVector<ValType> > {
    public:
      TMatrix(int s=10);                              //                    (#О1)
      TMatrix(const TMatrix &mt);                     // копирование        (#Л1)
      TMatrix(const TVector<TVector<ValType> > &mt);  // преобразование типа (#Л2)
      TMatrix & operator==(const TMatrix &mt);        // сравнение          (#П1)
      TMatrix & operator= (const TMatrix &mt);        // присваивание       (#О2)
      TMatrix   operator+ (const TMatrix &mt);        // сложение           (#П2)
      TMatrix   operator- (const TMatrix &mt);        // вычитание          (#С1)
      TMatrix   operator* (const TMatrix &mt);        // умножение          (#С2)
      // ввод / вывод
      friend istream & operator>>( istream &in, TMatrix &mt) {          //(#П3)
        for ( int i=0; i<mt.Size; i++ )                          // SKIP_ON
          in >> mt.pVector[i];                                   // SKIP_OFF
        return in;
      }
      friend ostream & operator<<( ostream &out, const TMatrix &mt) {   //(#С3)
        for ( int i=0; i<mt.Size; i++ )                          // SKIP_ON
          out << mt.pVector[i] << endl;                          // SKIP_OFF
        return out;
      }
};

template <class ValType>                  //
TMatrix<ValType> :: TMatrix ( int s ) : TVector<TVector<ValType> >(s) {
  for ( int i=0; i < s; i++ )
    pVector[i] = TVector<ValType>(s-i,i);
}

template <class ValType>                  // конструктор копирования
TMatrix<ValType> :: TMatrix ( const TMatrix<ValType> &mt ) :
  TVector<TVector<ValType> >(mt) {}                            // SKIP_ON SKIP_OFF
```

```
template <class ValType>                    // конструктор преобразования типа
TMatrix<ValType> :: TMatrix ( const TVector<TVector<ValType> > &mt ) :
  TVector<TVector<ValType> >(mt) {}                              // SKIP_ON SKIP_OFF

template <class ValType>                    // присваивание
TMatrix<ValType> & TMatrix<ValType>::operator=(const TMatrix<ValType> &mt){
  if ( this != &mt ) {
    if ( Size != mt.Size ) {
      delete[] pVector;
      pVector = new TVector<ValType>[mt.Size];
    }
    Size = mt.Size; StartIndex = mt.StartIndex;
    for ( int i=0; i<Size; i++ ) pVector[i] = mt.pVector[i];
  }
  return *this;
}

template <class ValType>                    // сложение
TMatrix<ValType> TMatrix<ValType>::operator+(const TMatrix<ValType> &mt) {
  return TVector<TVector<ValType> >::operator+(mt);              // SKIP_ON SKIP_OFF
}
// TVector О3 Л2 П4 С6
// TMatrix О2 Л2 П3 С3
#endif

//-------------------------------------------------------------------------
```

```
// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// utmatr4.h - Copyright (c) Гергель В.П. 07.05.2001
//
// Верхние треугольные матрицы - тест

#include <iostream.h>
#include <conio.h>
#include "utmatr4.h"
//---------------------------------------------------------------------------

#pragma argsused
int main(int argc, char* argv[]) {
  TMatrix<int> a(5), b(5), c(5);
  int i, j;
  cout << "Тестирование программ поддержки треугольных матриц" << endl;
  for ( i=0; i<5; i++ )
    for ( j=i; j<5; j++ ) {
      a[i][j] =  i*10 + j;
      b[i][j] = (i*10 + j)*100;
  }
  c = a + b;
  cout << "Matrix a = " << endl << a << endl;
  cout << "Matrix b = " << endl << b << endl;
  cout << "Matrix c = a + b" << endl << c << endl;
  cout << "Нажмите любую клавишу" << endl;
  getch();
  return 0;
}
//---------------------------------------------------------------------------
```

Нижегородский Государственный Университет им. Н.И. Лобачевского

Общий курс:

Методы программирования - 2

Тема 3:

Динамические структуры данных

Гергель В.П., профессор
кафедра МО ЭВМ ВМК

Содержание

1.3. Динамические структуры данных

1. Преобразование структур данных в процессе вычислений...

☑ Структуры данных являются операндами операций обработки

☑ Результаты вычислений также являются структурами, модель которых может как совпадать, так и отличаться от структуры исходных данных

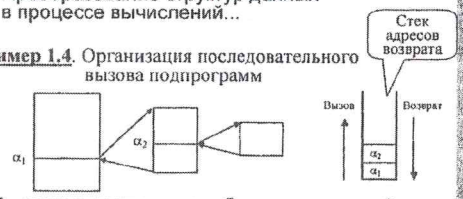
Пример: Произведение векторов

$$ab^T = \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix} * (b_1 \dots b_m) = \begin{pmatrix} c_{11} \dots c_{1m} \\ \dots \\ c_{n1} \dots c_{nm} \end{pmatrix} = c$$

1.3. Динамические структуры данных

1. Преобразование структур данных в процессе вычислений...

Пример 1.4. Организация последовательного вызова подпрограмм



☑ Для возврата в точку вызова необходимо запоминать *адрес возврата*

☑ При завершении вызываемой подпрограммы для возврата используется последний запомненный адрес

☑ В ходе вызова подпрограмм количество запоминаемых адресов постоянно изменяется (увеличивается при вызове очередной подпрограммы и уменьшается после завершения работы текущей подпрограммы)

1.3. Динамические структуры данных

1. Преобразование структур данных в процессе вычислений...

☑ Отличительная особенность – структура исходных и результирующих данных являются близкими

Выполним анализ рассмотренного примера

- Текущий набор адресов – линейная структура $S_n=(a_1 a_2 \dots a_n)$
- Пусть T – операция исключения последнего адреса. Тогда

$$T(a_1 a_2 \dots a_n a_{n+1}) = (a_1 a_2 \dots a_n)$$

- Пусть P – операция добавления нового адреса. Тогда

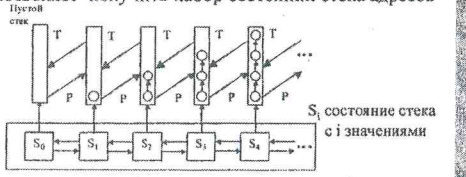
$$P(a_{n+1}; (a_1 a_2 \dots a_n)) = (a_1 a_2 \dots a_n a_{n+1})$$

☑ Орграф результата является подорграфом орграфа операнда или включает его

1.3. Динамические структуры данных

1. Преобразование структур данных в процессе вычислений...

☑ Последовательное применение операций T и P позволяет получить набор состояний стека адресов



☑ Мы имеем множество $\mathcal{M}$ элементов, у которых есть имена и значения

☑ $S_i \subset S_{i+1}$, т.е. элементы связаны отношением включения подграфов

1. Преобразование структур данных в процессе вычислений

Пусть:
- $\mathcal{P}_1$ - отношение следования, порождаемое операцией вставки,
- $\mathcal{P}_2$ - отношение следования, порождаемое операцией исключения

Тогда *стек* есть структура

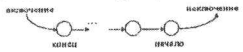
$$S=(\mathcal{M}, \mathcal{P}_1, \mathcal{P}_2),$$

в которой
- каждый элемент – структура,
- в любой момент существует только один конкретный элемент из $\mathcal{M}$,
- элементы частично упорядочены по включению.

2. Понятие динамической структуры

Определение 1.9. *Динамическая структура* есть математическая структура, которой соответствует частично-упорядоченное (по включению) базовое множество $\mathcal{M}$, элементы которого являются структурами данных. При этом отношения включения индуцируются операциями преобразования структуры данных.

Пример 1.5. Очередь (вставка в конец очереди, исключение из начала – дисциплина FIFO – first in, first out)



Пример 1.6. Дек (dequeue – double ended queue) – вставка и исключение для начала и конца дека - дисциплина FOLIFOLO – first or last in, first or last out)

1. Общая структура хранения элементов и программы реализации отношения включения…

- ☑ В каждый текущий момент времени в памяти хранится только один элемент базисного множества
- ☑ Для преобразования текущего элемента к следующему должны быть разработаны *программы поддержки* динамической структуры
- ☑ Структура хранения текущего элемента должна иметь достаточно общий характер, достаточный для представления любых элементов базисного множества структуры

1. Пример разработки структуры хранения

Практическая работа 3: Реализация стека

2. Сравнение линейных и динамических структур

	Линейные структуры	Динамические структуры
1	Базисное множество – множество элементов	Элементы базисного множества являются структурами ("ДС – структуры структур")
2	Базисное отношение – отношение следования	Базисное отношение – отношение включения
3	В структуре хранения хранятся все элементы структуры	В структуре хранения хранится только текущий элемент структуры
4	Отношение следования реализуется при помощи адресной арифметики	Отношение включения реализуется при помощи программ

Определение1.10. Программы, реализующие отношение включения, называются *средствами поддержания динамической структуры*

3. Проблема эффективного использования памяти…

- ☑ При использовании стека может возникнуть ситуация исчерпания памяти для хранения значений – *ограничение реализации* теоретически неограниченного стека
- ☑ Невозможность использования стека из-за переполнения возникает только в момент полного использования всей выделенной для стека памяти

Определение 1.11. Для оценки эффективности использования памяти введем показатель E_{mem}, определяемый как отношение объема используемой памяти к общему размеру выделенной памяти.

1.4. Динамические структуры и структуры хранения

3. Проблема эффективного использования памяти

Практическая работа 4: Реализация очереди

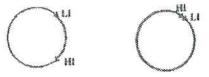
1.4. Динамические структуры и структуры хранения

4. Использование нескольких структур и необходимость динамического распределения памяти

Пример 1.7. Стек для хранения фиксированного количества последних записанных значений

Стек подобного вида может быть использован, например, для запоминания данных, достаточных для восстановления текущего состояния программы перед выполнением очередной операции обработки (для реализации процедуры типа операции отмены в редакторе текстов). При реализации данного механизма обычно фиксируется максимальная глубина отката; при заполнении памяти из стека удается наиболее старая записанная информация.

- Наиболее простой способ реализации такого стека – использование кольцевого буфера (при заполнении стека следующий для использования по кольцу элемент памяти содержит подлежащее удалению значение данных)



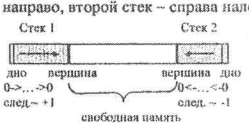
☑ Как реализовать процедуру отмены операции отката ?

1.4. Динамические структуры и структуры хранения

4. Использование нескольких структур и необходимость динамического распределения памяти

Пример 1.8. Задача использования двух стеков

- ☑ При использовании двух стеков может возникнуть ситуация переполнения одного стека, в то время как в другом стеке свободная память может еще присутствовать – эффективность использования памяти в этом случае не будет являться максимально-возможной
- Возможное решение проблемы – использование общей памяти для этих стеков, в которой один стек будет "расти" слева направо, второй стек – справа налево.



☑ При таком подходе свободная память является общей и ситуация переполнения стеков возникает только при полном исчерпании памяти

1.4. Динамические структуры и структуры хранения

4. Использование нескольких структур и необходимость динамического распределения памяти

Определение 1.13. Распределение памяти до начала процесса вычислений называется *статическим*. Распределение памяти в ходе выполнения программы называется *динамическим распределением памяти*.

☑ Как распределить память для большего, чем 2, количества стеков ?

Заключение

- Понятие динамической структуры данных
- Необходимость разработки общей структуры хранения для элементов базисного множества
- Необходимость программной реализации отношения включения
- Отличительные особенности линейных и динамических структур
- Структура хранения в виде кольцевого буфера
- Статическое и динамическое распределение памяти
- Примеры реализации динамических структур: стеки и очереди
- Программирование с защитой от ошибок

Вопросы для обсуждения

- Классификация видов структур данных: линейные, динамические…
- Проблема разных типов значений для элементов базисных множеств
- Принципы программирования с защитой от ошибок (накладные расходы, увеличение объема программирования, существующие подходы…)
- Достоинства и недостатки динамического распределения памяти

Темы заданий для самостоятельной работы

- Реализация стеков и очередей с использованием шаблонов
- Разработка программы для вычисления арифметического выражения в постфиксной записи

Следующая тема

- Динамическое распределение памяти

Нижегородский Государственный Университет им. Н.И. Лобачевского

Общий курс:

Методы программирования - 2

Практическая работа 3:

Структура хранения стека

Гергель В.П., профессор
кафедра МО ЭВМ ВМК

Содержание

1. Разработка спецификаций...

- S - стек, $|S|$ - количество элементов в стеке
- S_n^k - стек, n – размер памяти, k – количество значений,
- δ - код выполнения операции (=1 – стек пуст, =2 – стек полон)

Операции
- $S(n) \rightarrow S_n^0$ - создание стека
- $S<<x \rightarrow S'$ - вставка в стека (при $k=n \Rightarrow S' = S$ и $\delta=2$)
- $S>>x \rightarrow S'$ - исключение из стека (при $k=0 \Rightarrow S' = S$ и $\delta=1$)
- $\alpha_0(S)$ - предикат проверки пустоты стека ($\alpha_0(S)=1$ при k=0)
- $\alpha(S)$ - предикат проверки переполнения памяти ($\alpha(S)=1$ при k=n)

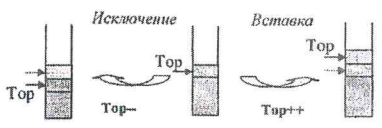
Аксиомы
- $S=S(n) \Rightarrow S = S_n^0$ и $\alpha_0(S)=1$ – созданный стек является пустым
- $S'=S<<x \Rightarrow \alpha_0(S)=0$ – стек, в который выполнена вставка, не пуст
- $S'=S>>x \Rightarrow \alpha(S)=0$ – стек после операции исключения не полон
- $S'=(S<<x)>>y \Rightarrow x=y$ – при выборке значения из стека извлекается последнее вставленное значение
- $S'=S>>x$, $\alpha_0(S)=1 \Rightarrow S' = S$ и $\delta=1$ – выборка значения из пустого стека устанавливает код завершения $\delta=1$
- $S'=S<<x$, $\alpha(S)=1 \Rightarrow S' = S$ и $\delta=2$ – вставка значения в полный стек устанавливает код завершения $\delta=2$

1. Разработка спецификаций

Тесты
- общий (вставка значений определенной последовательности значений и проверка при выборке)
- программы проверки выполнения аксиом (регулярное или периодическое выполнение)
- проверка исключительных ситуаций (пустота, переполнение)
- визуализация структуры или содержимого стека
- печать значений стека
- копирование значений стека (в массив или файл)
- проверка корректности структуры хранения

2.1. Начальный вариант реализации – структура хранения

- ☑ Для хранения элементов базисного множества выделяется вектор памяти размера, достаточного для хранения максимально-возможного количества (!?) значения в стеке
- ☑ Хранение значений в памяти осуществляется последовательно от младшего элемента вектора к старшему
- ☑ Для запоминания количества хранимых в стеке значений используется индекс последнего занятого элемента в векторе



2.2 Начальный вариант реализации – операции

- IsEmpty – проверка пустоты
- IsFull – проверка переполнения
- Put – добавить значение
- Get – извлечь значение

Программа

Структура хранения стека

3.1. Расширенный вариант реализации – контроль ошибок

Программирование с защитой от ошибок

☑ При разработке программного кода проверяется выполнимость всех условий, которые априори должны иметь место (допустимость значений переменных, истинность тех или предикатов и т.п.). При обнаружении ошибок:
- аварийное завершение программы (с той или иной диагностикой)
- формирование специальных кодов завершения, которые могут быть проанализированы на более высоком уровне управления

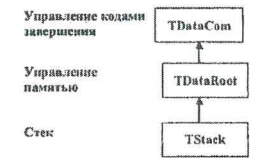
☞ Для управления кодами завершения может быть разработан общий базовый класс TDataCom
- при завершении любого метода производного класса должна вызываться процедура SetRetCode(int RetCode)
- для получения кода завершения последней выполненной программы используется метод int GetRetCode()

Программа

Структура хранения стека

3.2. Расширенный вариант реализации – схема наследования

☑ Реализацию стека можно разделить на 2 этапа:
- разработка абстрактного базового класса TDataRoot для управления памятью и определения спецификаций необходимых операций
- разработка класса TStack, обеспечивающего представления стека



Контрольный пример: программа, приложение

Структура хранения стека

3.2. Расширенный вариант реализации – схема наследования

☑ Класс TDataRoot является абстрактным базовым классом и обеспечивает управление памятью и определяет спецификацию операций, которые должны обеспечиваться структурой данных:

Управление памятью – память для хранения элементов структуры может выделяться двумя различными способами (использованный вариант фиксируется в переменной MemType):
- память выделяется конструктором из динамически-распределяемой области (MemType=MEM_HOLDER)
- память передается объекту при помощи метода SetMem (MemType=MEM_RENTER)

Поля
- pMem - указатель на память для хранения значений
- MemSize – размер памяти
- DataCount – количество значений, хранимых в структуре

Методы
- IsEmpty – проверка пустоты структуры
- IsFull – проверка переполнения памяти
- Put – операция вставки нового значения в структуру
- Get – операция выборки очередного значения из структуры

Структура хранения стека

3.2. Расширенный вариант реализации – схема наследования

☑ Класс TStack обеспечивает реализацию динамической структуры стек:

Управление памятью – хранение значений осуществляется в векторе памяти от младших элементов к старших; индекс последнего занятого элемента в векторе памяти фиксируется в переменной Hi

Методы
- GetNextIndex – метод для получения следующего значения индекса (скрытие способа реализации отношения следования)
- Put – операция вставки нового значения в структуру
- Get – операция выборки очередного значения из структуры

Заключение

- Разработка спецификаций
- Выбор структуры хранения стека
- Программирование с защитой от ошибок
- Определение схемы наследования

Вопросы для обсуждения

- Управление памятью в структуре хранения стека (выделение и передача памяти)
- Реализация отношения следования в виде виртуального метода

Темы заданий для самостоятельной работы

- Реализация тестов
- Разработка контрольного примера

Следующая тема

- Структура хранения очереди

```
// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// tstack.h - Copyright (c) Гергель В.П. 10.10.2001
//
// Динамические структуры данных - простая реализация стека

#ifndef __TSTACK_H
#define __TSTACK_H

#define MemSize   25  // размер памяти для стека

class TStack {
  protected:  // поля
    int Mem[MemSize];   // память для СД
    int Top;            // индекс последнего занятого в Mem
  public:
    TStack () { Top = -1; }
    int IsEmpty ( void ) const  { Top == -1;          } // контроль пустоты
    int IsFull  ( void ) const  { Top == MemSize-1;   } // контроль переполнения
    void  Put ( const int Val ) { Mem[++Top] = Val;   } // добавить значение
    TData Get ( void )          { return Mem[Top--]; } // извлечь значение
};
#endif
```

```
// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// datacom.h - Copyright (c) Гергель В.П. 30.08.2000
//
// Обработка кодов завершения

#ifndef __DATACOM_H
#define __DATACOM_H

#define DataOK   0
#define DataErr -1

class TDataCom {
 protected:
  int  RetCode; // Код завершения
  int  SetRetCode(int ret) { return RetCode=ret; };
 public:
  TDataCom () : RetCode(DataOK) {};
  virtual ~TDataCom() {};
  int  GetRetCode()   {
    int temp=RetCode; RetCode=DataOK; return temp;
  };
};
/* Notes:
     TDataCom является общим базовым классом
*/
#endif
```

```
// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// dataroot.h - Copyright (c) Гергель В.П. 28.07.2000 (06.08)
//
// Динамические структуры данных - базовый (абстрактный) класс - версия 3.2
//   память выделяется динамически или задается методом SetMem

#ifndef __DATAROOT_H
#define __DATAROOT_H

#include "datacom.h"

#define DefMemSize   25  // размер памяти по умолчанию

#define DataEmpty  -101  // СД пуста
#define DataFull   -102  // СД переполнена
#define DataNoMem  -103  // нет памяти

typedef int    TElem;    // тип элемента СД
typedef TElem *PTElem;
typedef int    TData;    // тип значений в СД

enum TMemType { MEM_HOLDER, MEM_RENTER };

class TDataRoot : public TDataCom {
  protected:  // поля
    PTElem pMem;       // память для СД
    int MemSize;       // размер памяти для СД
    int DataCount;     // к-во элементов в СД
    TMemType MemType; // режим управления памятью
  protected:  // методы
    void SetMem ( void *pMem, int Size );       // задание памяти
  public:
   ~TDataRoot ();
    TDataRoot ( int Size=DefMemSize );
    virtual int IsEmpty ( void ) const;         // контроль пустоты СД
    virtual int IsFull  ( void ) const;         // контроль переполнения СД
    virtual void  Put   ( const TData &Val )=0; // добавить значение
    virtual TData Get   ( void )            =0; // извлечь значение
    // служебные методы
    virtual void Print()    = 0;                // печать значений
    virtual int  IsValid() = 0;                 // тестирование структуры
    // дружественные классы
    friend class TMultiStack;
    friend class TSuperMultiStack;
    friend class TComplexMultiStack;
};
#endif
```

```
// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// dataroot.cpp - Copyright (c) Гергель В.П. 28.07.2000 (06.08)
//
// Динамические структуры данных - базовый (абстрактный) класс - версия 3.1
//   память выделяется динамически или задается методом SetMem

#include <alloc.h>
#include <stdio.h>
#include "dataroot.h"

      /*-------------------------------------------*/
TDataRoot :: TDataRoot ( int Size ) : TDataCom() {
  DataCount = 0;
  MemSize   = Size;
  if ( Size == 0 ) { // память будет установлена методом SetMem
    pMem = NULL; MemType = MEM_RENTER;
  }
  else { // память выделяется объектом
    pMem  = new TElem[MemSize];
    MemType = MEM_HOLDER;
  }
}
      /*-------------------------------------------*/

TDataRoot :: ~TDataRoot () {
  if ( MemType == MEM_HOLDER ) delete pMem;
  pMem = NULL;
}
      /*-------------------------------------------*/

void TDataRoot :: SetMem ( void *pMem, int Size ) { // задание памяти
  if ( MemType == MEM_HOLDER ) delete pMem; // ! информация не сохраняется
  pMem    = (TElem *) pMem;
  MemType = MEM_RENTER;
  MemSize = Size;
} // SetMem
      /*-------------------------------------------*/


int TDataRoot :: IsEmpty(void) const { // контроль пустоты СД
  return DataCount == 0;
}
            /*-------------------------------------------*/

int TDataRoot :: IsFull(void) const { // контроль переполнения СД
  return DataCount == MemSize;
}
```

```
// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// Copyright (c) Гергель В.П. 28.07.2000 (07.08)
//
// Динамические структуры данных - стек

#ifndef __DATSTACK_H
#define __DATSTACK_H

#include "dataroot.h"

#define StackID  101

class TStack : public TDataRoot {
  protected:
    int Hi;                                  // индекс последнего элемента структуры
    virtual int GetNextIndex(int index); // получить следующий индекс
  public:
    TStack ( int Size=DefMemSize ): TDataRoot(Size) { Hi=-1; }
    virtual void  Put ( const TData &Val );  // положить в стек (#Л1)
    virtual TData Get ( void );              // взять из стека с удалением
  protected:
    // служебные методы
    virtual void Paint(int y,int x1,int x2); // показать рисунок структуры
    virtual int  IsValid();                  // тестирование структуры
    virtual void Print();                    // печать значений
    virtual void CopyToVector(TElem v[]);    // копировать в очередь
};
#endif

// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// Copyright (c) Гергель В.П. 27.07.2000 (07.08)
//
// Динамические структуры данных - стек - версия 3.1

#include <stdio.h>
#include <conio.h>
#include "datstack.h"

void TStack :: Put ( const TData &Val ) { // положить в стек
  if ( pMem == NULL ) SetRetCode ( DataNoMem );   // SKIP_ON
  else if ( IsFull() ) SetRetCode ( DataFull  );
  else {
    Hi = GetNextIndex(Hi);
    pMem[Hi] = Val;
    DataCount++;
  }                                                // SKIP_OFF
} // Put
/*-------------------------------------------*/

TData TStack :: Get(void) { // взять из стека с удалением
  TData temp = -1;
  if ( pMem == NULL  ) SetRetCode ( DataNoMem );
  else if ( IsEmpty() ) SetRetCode ( DataEmpty );
  else {
    temp = pMem[Hi--];
    DataCount--;
  }
  return temp;
} // Get
/*-------------------------------------------*/

int TStack :: GetNextIndex(int index) { // получить следующее значение  индекса
  return ++index;
} // GetNextIndex
```

```
void TStack :: Paint(int y,int x1,int x2) { // показать рисунок структуры
  int i, px;
  gotoxy(x1,y);
  if ( DataCount == 0 ) px = x1 - 1;
  else {
    px = x1 + double(DataCount) * (x2-x1+1) / MemSize - 1;
    if ( (px<x1) && (DataCount>0) ) px = x1;
    if ( DataCount == MemSize )     px = x2;
  }
  for ( i=x1;   i<=px; i++ ) cprintf("#");
  for ( i=px+1; i<=x2; i++ ) cprintf(" ");
  if ( (x1<1)||(x1>79) || (px<1)||(px>79) || (x2<1)||(x2>79) ) {
    printf("Error in parameters of painting...\n");
//    getch();
  }
}
/*-----------------------------------------------*/

void TStack :: Print() { // печать значений стека
  for ( int i=0;   i<DataCount; i++ )
    printf("%d ",pMem[i]);
  printf("\n");
}
/*-----------------------------------------------*/

int  TStack :: IsValid() { // тестирование структуры
  int res = 0;
  if ( pMem == NULL )                    res  = 1;
  if ( MemSize < DataCount )             res += 2;
  return res;
}
/*-----------------------------------------------*/

void TStack :: CopyToVector(TElem v[]) { // копировать в очередь
  for ( int i=0; i<DataCount; i++ )
    v[i] =pMem[i];
}
```

Нижегородский Государственный Университет им. Н.И. Лобачевского

Общий курс:

Методы программирования - 2

Практическая работа 4:

Структура хранения очереди

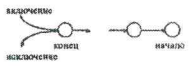
Гергель В.П., профессор
кафедра МО ЭВМ ВМК

Содержание

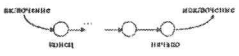
Структура хранения очереди

1. Проблема эффективного использования памяти…

Стек (вставка и выборка значений для вершины стека – дисциплина FILO – first in, last out)



Очередь (вставка в конец очереди, исключение из начала – дисциплина FIFO – first in, first out)

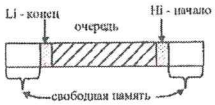


Структура хранения очереди

1. Проблема эффективного использования памяти

Очередь

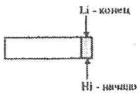
☑ Вставка значений происходит в начало очереди, исключение значений – с конца очереди ⇒ для индикации начала и конца очереди требуется два индекса



Выборка из элемента с индексом Li и увеличение Li

Вставка - увеличение Hi и запись в элемент с индексом Hi

В ходе вычислений может возникнуть ситуация Li= Hi=n-1. Тогда вставка нового значения невозможна, а $E_{mem} \approx 0$ (!)

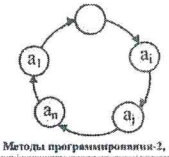


Структура хранения очереди

2. Способы достижения полного использования памяти…

- сдвиг значений очереди после выборки очередного значения (т.е. обеспечение Li=0) – возрастание накладных расходов
- использование левого участка свободной области при достижении Hi=n-1 (т.е. при отсутствии свободной памяти справа)

Определение 1.12. Структура хранения, получаемая из вектора расширением отношения следования парой $p(a_n,a_1)$, называется *циклическим или кольцевым буфером*.

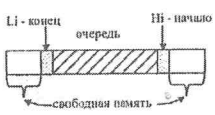


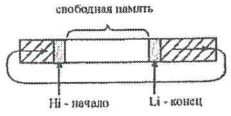
Структура хранения очереди

2. Способы достижения полного использования памяти…

☑ Реализация кольцевого буфера логически может быть обеспечена переходом индексов Li и Hi при достижении граничного значения MemSize-1 на индекс первого элемента вектора памяти

Ситуации для структуры хранения очереди



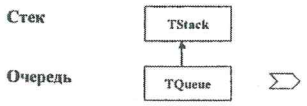


Структура хранения очереди

3. Схема наследования и реализация очереди

☑ Какой базовый класс взять для реализации ? TStack

☑ Какие методы переопределить ? GetNextIntex
Put



Контрольный пример: программа, приложение

Заключение

- Проблема эффективного использования памяти
- Способы достижения полного использования памяти
- Структура хранения в виде кольцевого буфера
- Схема наследования и реализация очереди

Вопросы для обсуждения

- Разработка класса очереди TQueue через наследование класса стека TStack
- Реализация отношения следования в виде виртуального метода

Темы заданий для самостоятельной работы

- Разработка приоритетной очереди
- Реализация дека

Следующая тема

- Структура хранения нескольких динамических структур данных

```
// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// Copyright (c) Гергель В.П. 28.07.2000 (07.08)
//
// Динамические структуры данных - очередь - версия 3.1

#ifndef __DATQUEUE_H
#define __DATQUEUE_H

#include "datstack.h"
#define QueueID  102

class TQueue : public TStack {
  protected:
    int Li;                                  // индекс первого элемента структуры
    virtual int GetNextIndex(int index); // получить следующий индекс
  public:
    TQueue(int Size=DefMemSize) : TStack(Size) { Li=0; }
    virtual TData Get (void); // взять из очереди с удалением (#Л1)
};
#endif
```

```
// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// Copyright (c) Гергель В.П. 28.07.2000
//
// Динамические структуры данных - очередь - версия 3.1

#include "datqueue.h"

int TQueue :: GetNextIndex(int index) { // получить следующее значение  индекса
  return ++index % MemSize;             // логическое представление кольцевого
буфера
} // GetNextIndex
/*-------------------------------------------*/

TData TQueue :: Get(void) { // взять из очереди с удалением
  TData temp = -1;                                  // SKIP_ON
  if ( pMem == NULL  ) SetRetCode ( DataNoMem );
  else if ( IsEmpty() ) SetRetCode ( DataEmpty );
  else {
    temp = pMem[Li];
    Li = GetNextIndex(Li);
    DataCount--;
  }
  return temp;                                      // SKIP_OFF
} // Get
```

```
// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// Copyright (c) Гергель В.П. 28.07.2000 (07.08)
//
// Динамические структуры данных - очередь - тест

#include "dataroot.cpp"
#include "datstack.cpp"
#include "datqueue.cpp"
#include <iostream.h>
#include <conio.h>

int main(int argc, char* argv[]) {
  TQueue st;
  int temp;
  cout << "Тестирование программ поддержки структуры типа очереди" << endl;
  for ( int i=0; i<35; i++ ) {
    st.Put(i);
    cout << "Положили значение " << i << " Код " << st.GetRetCode() << endl;
  }
  getch();
  int k;
  while ( !st.IsEmpty() ) {
    temp = st.Get();
    cout << "Взяли значение " << temp << " Код " << st.GetRetCode() << endl;
  }
  cout << "Нажмите любую клавишу" << endl;
  getch();
  return 0;
}
```

Нижегородский Государственный Университет им. Н.И. Лобачевского

Общий курс:

Методы программирования - 2

Тема 4:

Динамическое распределение памяти

Гергель В.П., профессор
кафедра МО ЭВМ ВМК

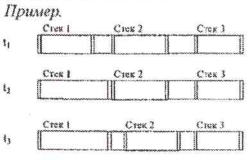
Содержание

1.5. Динамическое распределение памяти

1. Динамическое распределение памяти...

Определение 1.13. Распределение памяти до начала процесса вычислений называется *статическим*. Распределение памяти в ходе выполнения программы называется *динамическим распределением памяти*.

Пример.



В общей области памяти располагаются 3 стека

Стек 1 переполнен

Перераспределение свободной памяти путем перемещения стеков по памяти

Определение 1.14. Процедура динамического перераспределения памяти путем переписи части хранимых значений в другую область памяти называется *перепаковкой памяти* или просто *перепаковкой*.

1.5. Динамическое распределение памяти

1. Динамическое распределение памяти

☑ Перепаковка обеспечивает эффективное использование одного ресурса ЭВМ (*памяти*) за счет другого ресурса (*времени*).

☑ Для выполнения перепаковки требуется разработка управляющих программ.

Определение 1.15. Выполнение функций анализа свободной памяти, планирование размещения структур, переписывание структур называется *управление памятью*. Комплекс программ, реализующих управление памятью называется *системой управления памятью*.

☑ Необходимость перепаковки обуславливается принятым способом реализации отношений следования.

1.5. Динамическое распределение памяти

2. Динамическое распределение путем перепаковки структур хранения

Практическая работа 5: Разработка системы поддержки нескольких стеков

1.5. Динамическое распределение памяти

3. Роль гипотез о поведении стеков...

☑ Гипотезы о поведении структур служат основой для принятия решений о распределении памяти

☑ Формирование гипотез происходит в результате теоретического анализа модели решаемой задачи или может быть выполнено на основе статистических данных, получаемых в ходе вычислительных экспериментов с проектируемой программной системой

- **Гипотеза 1**: Стеки используются с одинаковой интенсивностью ⇒ память разделяется между стеками поровну
- **Гипотеза 2**: Интенсивность использования стеков различается. Конструктивное предположение о характере такой неравномерности может состоять в гипотезе *сохранения локальных тенденций роста* стеков, т.е. в каждый момент времени использование стеков на последующих шагах вычислений характеризуется точно таким же поведением, что и на предшествующих этапах обработки данных.

1.5. Динамическое распределение памяти

3. Роль гипотез о поведении стеков...

- **Гипотеза 2**: Сохранение локальных тенденций роста
  - показатель роста стека
    $$\delta_i = \max (0, DataCount'_i - DataCount_i)$$
  - суммарный показатель роста
    $$\Delta = \Sigma \delta_i,\ 0 \le i < N$$
  - правило распределение памяти для стеков в соответствии с их показателями роста
    $$Li'_k = Li'_{k-1} + (Hi_{k-1} - Li_{k-1} + 1) + F*(\delta_i / \Delta),\ 1 \le k < N$$
- Как изменить программы системы для применения нового варианта перераспределения памяти ?
- Достаточно переопределить метод планирования памяти для перепаковки SetStackLocation (процедуру оценки показателей роста целесообразно выделить в отдельный метод SetStackRate)
- Контрольный пример: программа, приложение

1.5. Динамическое распределение памяти

3. Роль гипотез о поведении стеков...

- **Гипотеза 3**: Использование вероятностных предположений о поведении стеков

Пусть есть θ, $0 \le \theta \le 1$, вероятность выполнимости гипотезы сохранения локальных тенденций роста. Тогда

$$Li'_k = Li'_{k-1} + (Hi_{k-1} - Li_{k-1} + 1) + (1-\theta)*(F/N) + \theta*F*(\delta_i / \Delta),\ 1 \le k < N$$

Распределение памяти поровну между стеками

Распределение памяти пропорционально показателям роста

- Какая дополнительная доработка системы требуется для применения новой схемы распределения памяти ?
- Достаточно переопределить метод планирования памяти для перепаковки SetStackLocation (!)
- Контрольный пример: программа, приложение

1.5. Динамическое распределение памяти

3. Роль гипотез о поведении стеков...

- **Адаптивная оценка параметров модели...**
- Как оценить вероятность выполнимости гипотезы сохранения локальных тенденций роста θ ?
- Пусть σ есть количество выполненных перепаковок памяти за некоторый отрезок времени Δt. Величина σ зависит от значения θ и для повышения эффективности функционирования системы следует определить такое θ, чтобы количество перепаковок было минимальным, т.е.

$$\min_{\theta} \sigma(\theta)$$

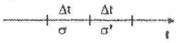
1.5. Динамическое распределение памяти

3. Роль гипотез о поведении стеков...

- **Адаптивная оценка параметров модели...**

$$\min_{\theta} \sigma(\theta)$$

- Возможная схема определения оптимального значения θ может состоять в следующем:
  - выполним оценку величины σ на последовательных друг за другом отрезках времени Δt



  и определим величину изменения количества выполненных перепаковок:

$$\Delta\sigma = \sigma' - \sigma$$

  - примем следующее следующее правило корректировки значения θ

$$\theta' = \begin{cases} \theta + \Delta\theta, & \Delta\sigma \le 0, \\ \theta - \Delta\theta, & \Delta\sigma > 0, \end{cases}$$

где $\Delta\theta$ есть параметр схемы адаптации

1.5. Динамическое распределение памяти

3. Роль гипотез о поведении стеков...

- **Адаптивная оценка параметров модели...**
- Для включения в разработанную программную системы схемы адаптации по прежнему достаточно переопределить метод планирования памяти перепаковки SetStackLocation (!)
- Контрольный пример: программа, приложение

Результаты вычислительных экспериментов

Гипотеза	Итераций	Количество перепаковок	Среднее	
Одинаковое использование	2007	1018	1, 97	
Сохранение тенденций роста	1195	726	1, 64	
Смешанная	2578	1337	1, 92	θ=0.5
Адаптация	3031	1474	2, 06	θ=0.3

Заключение

- Статическое и динамическое распределение памяти
- Система управления памяти
- Система поддержки нескольких стеков
- Роль гипотез о поведении структур данных
- Оценка параметров модели в ходе вычислений

Вопросы для обсуждения

- Динамическое распределение памяти путем перепаковки структуры хранения
- Гипотезы о поведении стеков
- Схемы адаптивного поведения программ
- Способы снижения затрат на организацию динамического распределения памяти

Темы заданий для самостоятельной работы

- Разработка системы поддержки нескольких очередей
- Моделирование динамики использования стеков

Следующая тема

- Структуры хранения с использованием сцепления (списки)

Нижегородский Государственный Университет им. Н.И. Лобачевского

Общий курс:

Методы программирования - 2

Практическая работа 5:

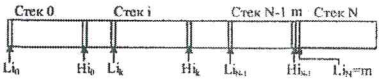
Структура хранения нескольких стеков в общей памяти

Гергель В.П., профессор
кафедра МО ЭВМ ВМК

Содержание

Разработка системы поддержки нескольких стеков

1. Структура памяти и ее свойства



где N – количество стеков, m – размер памяти

Свойства

- ☑ $Li_0=0$ - условие неподвижности 1 стека
- ☑ $Hi_k=Li_k-1$ - условие пустоты
- ☑ $Hi_k<Li_{k+1}$ - условие неперекрытия
- ☑ $Hi_k=Li_{k+1}-1$ - условие переполнения

Для выполнимости последних двух условий для всех стеков введем фиктивный стек N, для которого $Li_N=m$.

Разработка системы поддержки нескольких стеков

2. Начальное распределение памяти

Будем предполагать, что все стеки используются с одинаковой интенсивностью ⇒ память распределяется всем стекам поровну

- $Li_0=0$, $Li_N=m$
- $Li_k=Li_{k-1}+m/N$, $1 \le k < N$

Разработка системы поддержки нескольких стеков

3. Перепаковка - оценка свободной памяти

Выполняется при попытке вставки нового значения в стек s, у которого отсутствует свободная память

$F = \Sigma (Hi_k-Li_{k-1}-1)$, $1 \le k < N$

- ☑ F =0 – свободной памяти нет
- ☑ F =1 – свободен 1 элемент памяти и его следует отдать стеку s
- ☑ F >1 – необходимо перераспределить свободную память

Для гарантированного выделения свободной памяти стеку s при наличии только одного свободного элемента памяти (случай 2), выполним

- $Hi_s=Hi_s+1$ - перед началом процедуры оценки свободной памяти
- $Hi_s=Hi_s-1$ - после завершения перепаковки

Разработка системы поддержки нескольких стеков

4. Перепаковка - перераспределение свободной памяти

Снова предположим, что все стеки используются с одинаковой интенсивностью ⇒ свободная память должна распределиться всем стекам поровну

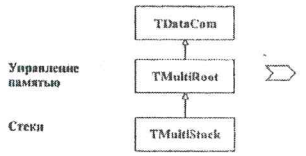
- $Li'_0= Li_0$, $Li'_N= Li_N$
- $Li'_k=Li'_{k-1}+(Hi_{k-1}-Li_{k-1}+1)+F/N$, $1 \le k < N$

☑ На равномерность распределения памяти может сказаться целочисленность операции деления

Разработка системы поддержки нескольких стеков

5.1. Реализация – схема наследования

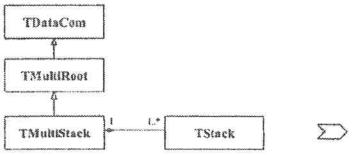
Реализацию системы можно снова разделить на 2 этапа:
- разработка абстрактного базового класса TMultiRoot для управления памятью и определения спецификаций необходимых операций
- разработка класса TMultiStack для обеспечения работы с несколькими стеками



Разработка системы поддержки нескольких стеков

5.2. Реализация – вставка и выборка значений

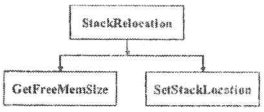
Для работы с отдельным стеком используется ранее разработанный класс TStack (!) – такая возможность обеспечивается ранее предусмотренной возможностью передачи извне памяти для хранения значений (метод SetMem)



Разработка системы поддержки нескольких стеков

5.3. Реализация – перепаковка…

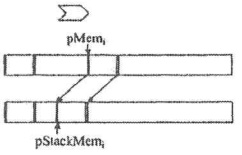
В соответствии с последовательностью действий, необходимых для выполнения перепаковки, определим следующий набор методов для динамического перераспределения памяти:
- StackRelocation – основный метод перепаковки;
- GetFreeMemSize – оценка свободной памяти;
- SetStackLocation – планирование нового расположения стеков (после перепаковки)
- GetRelocationCount – получение количества перепаковок за все время работы системы



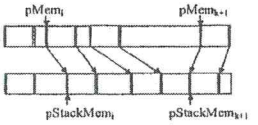
Разработка системы поддержки нескольких стеков

5.3. Реализация – перепаковка…

Перемещение стека влево



Перемещение стека вправо



Контрольный пример:
программа,
приложение

Заключение

- Структура памяти для размещения нескольких стеков
- Начальное распределение памяти
- Ситуация локального переполнения памяти
- Оценка свободной памяти
- Динамическое перераспределение памяти при помощи перепаковки данных
- Схема наследования и последовательность разработки программ
- Реализация перепаковки

Вопросы для обсуждения

- Стратегии динамического перераспределения памяти
- Модульность действий, выполняемых при перепаковке (оценка свободной памяти, стратегия перераспределения памяти,…)
- Использование ранее разработанного ПО (класс TStack)

Темы заданий для самостоятельной работы

- Разработка тестов для проверки правильности работы программ
- Динамическое изменение количества стеков

Следующая тема

- Гипотезы поведения стеков для определения стратегии динамического перераспределения памяти

```
// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// multroot.h - Copyright (c) Гергель В.П. 02.08.2000
//
// Динамические структуры данных - система N стеков
//   базовый абстрактный класс для наследования

#ifndef __MULTROOT_H
#define __MULTROOT_H

#include "datacom.h"

#define MemLimit  100  // размер памяти
#define StackNum   10  // количество стеков

class TMultiRoot : public TDataCom  {
  protected:
    TElem Mem[MemLimit]; // память для стеков
    int DefaultStack;    // номер текущего стека
  public:
    TMultiRoot () { DefaultStack = 0; }
    virtual int  IsEmpty ( int ns ) const = 0; // контроль пустоты СД
    virtual int  IsFull  ( int ns ) const = 0; // контроль переполнения СД
    virtual void Put     ( int ns, const TData &Val )=0; // положить в стек
    virtual TData Get    ( int ns ) = 0;       // взять из стека с удалением
// методы для работы с текущим стеком
    void SetDefaultStack ( int ns ){ DefaultStack = ns; } // текущий стек
    int  IsEmpty (void) const      { return IsEmpty(DefaultStack); } // пуст ?
    int  IsFull  (void) const      { return IsFull (DefaultStack); } // полон ?
    void Put ( const TData &Val )  { Put(DefaultStack,Val);    }     // в стек
    virtual TData Get (void)       { return Get(DefaultStack); }     // из стека
};
#endif

// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// Copyright (c) Гергель В.П. 30.07.2000
//
// Динамические структуры данных - система N стеков

#ifndef __MULTISTACK_H
#define __MULTISTACK_H

#include "datstack.h"
#include "multroot.h"

class TMultiStack : public TMultiRoot {
  protected:
    TStack *pStack[StackNum+1];      // стеки - память выделяется из StackMem
    int FreeMemSize;                 // размер свободной памяти
  protected:  // поля и методы для перепаковки
    TElem *pStackMem[StackNum+1];    // базовые адреса для памяти стеков
    int RelocationCount;             // количество перепаковок
    int StackRelocation ( int nst ); // перепаковка памяти  (#Л2)
    int GetFreeMemSize  ( void );    // оценка объема свободной памяти
    virtual void SetStackLocation ( TElem *pStackMem[] ); // оценка pStackMem
  public:
    TMultiStack ();
   ~TMultiStack ();
    int  IsEmpty ( int ns ) const;   // контроль пустоты СД
    int  IsFull  ( int ns ) const;   // контроль переполнения СД
    virtual void  Put ( int ns, const TData &Val ); // положить в стек (#Л1)
    virtual TData Get ( int ns );    // взять из стека с удалением
    int  GetRelocationCount() { return RelocationCount; }
```

```
// служебные методы
    void Paint(int y,int x1,int x2);  // показать рисунок структуры
    virtual int  IsValid();           // тестирование структуры
};
#endif

// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// Copyright (c) Гергель В.П. 31.07.2000
//
// Динамические структуры данных - система N стеков

#include <stdio.h>
#include <conio.h>
#include <string.h>
#include "multist.h"
#include "datqueue.h"

TMultiStack :: TMultiStack () {
  int StackSize = MemLimit / StackNum;
  for ( int i=0, pos=0; i<=StackNum; i++, pos+=StackSize ) {
    pStack[i] = new TStack(0);
    if ( i==StackNum-1 ) pStack[i]->SetMem ( &Mem[pos], MemLimit-pos );
    if ( i==StackNum   ) pStack[i]->SetMem ( &Mem[MemLimit], 0 );
    else pStack[i]->SetMem ( &Mem[pos], StackSize );
  }
  RelocationCount = 0;
  FreeMemSize = MemLimit;
  RetCode     = DataOK;
}
              /*-----------------------------------------------*/

TMultiStack :: ~TMultiStack () {
  for ( int i=0; i<=StackNum; i++ )
    delete pStack[i];
}
              /*-----------------------------------------------*/

int TMultiStack :: IsEmpty ( int ns ) const { // контроль пустоты СД
  return pStack[ns]->IsEmpty();
}
            /*-----------------------------------------------*/

int TMultiStack :: IsFull ( int ns ) const { // контроль переполнения СД
  return FreeMemSize==0;
}
            /*-----------------------------------------------*/

void TMultiStack :: Put ( int ns, const TData &Val ) { // положить в стек
  SetRetCode ( DataOK );                                // SKIP_ON
  if ( pStack[ns]->IsFull() ) {
    if ( ! StackRelocation(ns) ) SetRetCode ( DataNoMem );
  }
  if ( RetCode == DataOK ) {
    pStack[ns]->Put(Val);
    int  Code = pStack[ns]->GetRetCode();
    if ( Code == DataOK ) FreeMemSize--;
    SetRetCode ( Code );
  }                                                     // SKIP_OFF
} // Put
            /*-----------------------------------------------*/
```

```
TData TMultiStack :: Get ( int ns ) { // взять из стека с удалением
  TData temp = pStack[ns]->Get();
  int  Code = pStack[ns]->GetRetCode();
  if ( Code == DataOK ) FreeMemSize++;
  SetRetCode ( Code );
  return temp;
} // Get
            /*-------------------------------------------*/

int TMultiStack :: GetFreeMemSize (void) { // оценка объема свободной памяти
  FreeMemSize = 0;
  for ( int i=0; i<StackNum; i++ ) {
    FreeMemSize += pStack[i]->MemSize - pStack[i]->DataCount;
  }
  return FreeMemSize;
}
            /*-------------------------------------------*/
void TMultiStack :: SetStackLocation ( TElem *pStackMem[] ) {
// оценка новых значений для базовых адресов памяти стеков
// стратегия "всем поровну"
  pStackMem[0] = &Mem[0];
  for ( int i=1; i<StackNum; i++ )
    pStackMem[i] = pStackMem[i-1] + pStack[i-1]->DataCount + FreeMemSize /
StackNum;
  pStackMem[StackNum] = pStack[StackNum]->pMem;
}
            /*-------------------------------------------*/

int TMultiStack :: StackRelocation ( int nst ) { // перепаковка стеков
  int is, ns, ks, k, res=0;             // SKIP_ON
  pStack[nst]->DataCount++; // захват памяти для переполненного стека
  // оценка свободной памяти
  int temp = FreeMemSize;
  FreeMemSize = GetFreeMemSize();
  if ( FreeMemSize > -1 ) {
    res = 1; // свободная память есть - перепаковка
    RelocationCount++;
    // определение нового месторасположения стеков
    SetStackLocation ( pStackMem );
    // перемещение стеков
    for ( ns=0; ns<StackNum; ns++ )
      if ( pStackMem[ns] < pStack[ns]->pMem ) {      // смещение влево
        for ( k=0; k<pStack[ns]->DataCount; k++ )
          pStackMem[ns][k] = pStack[ns]->pMem[k];
        pStack[ns]->SetMem(pStackMem[ns],pStackMem[ns+1]-pStackMem[ns]);
      }
      else if ( pStackMem[ns] > pStack[ns]->pMem ) { // смещение вправо
        for ( ks=ns; pStackMem[ks+1] > pStack[ks+1]->pMem; ks++ );
        // стек (ks+1) -  первый справа, сдвигаемый влево
        for ( is=ks; is>=ns; is-- ) {           // сдвиг вправо стека is
          for ( k=pStack[is]->DataCount-1; k>=0; k-- )
            pStackMem[is][k] = pStack[is]->pMem[k];
          pStack[is]->SetMem(pStackMem[is],pStackMem[is+1]-pStackMem[is]);
        }
      }
      else // стек не перемещается - изменение размера
        pStack[ns]->SetMem(pStackMem[ns],pStackMem[ns+1]-pStackMem[ns]);
  }
  pStack[nst]->DataCount--; // возврат захваченной памяти
  FreeMemSize++;
  return res;                          // SKIP_OFF
}
            /*-------------------------------------------*/
```

```
void TMultiStack :: Paint(int y,int x1,int x2) { // показать рисунок структуры
  int FieldSize = ( x2 - x1 + 1 ) / StackNum;
  for ( int i=0, px=x1; i<StackNum; i++, px+=FieldSize )
    pStack[i]->Paint(y,px,px+FieldSize-1);
}
            /*---------------------------------------------*/
/*
void TMultiStack :: Print() { // печать значений стеков
  int i1, i2;
  for ( int i=0; i<StackNum; i++ ) {
    i1 = pStack[i]->pData-&Mem[0];
    i2 = pStack[i+1]->pData-&Mem[0]-1;
    printf("ns=%d, i1=%d, i2=%d, k=%d, m=%d -> ",
      i,i1,i2,pStack[i]->DataCount,pStack[i]->DataSize);
    pStack[i]->Print();
  }
}
            /*---------------------------------------------*/

int TMultiStack :: IsValid() { // тестирование структуры
  int res = 0, rc;
  for ( int i=0; i<StackNum; i++ ) {
    rc = pStack[i]->IsValid();
    if ( rc != 0 ) {
      printf("ns=%d, rc=%d\n",i,rc);
      res = 1;
    }
  }
  rc=0;
  for ( int i=0; i<StackNum; i++ ) {
    rc += (pStack[i+1]->pMem - pStack[i]->pMem);
  }
  if ( rc != MemLimit ) {
    printf("rc=%d\n",rc);
    res += 2;
  }
  rc=0;
  for ( int i=0; i<StackNum; i++ ) {
    rc += pStack[i]->MemSize;
  }
  if ( rc != MemLimit ) {
    printf("rc=%d\n",rc);
    res += 2;
  }
  rc = 0;
  if ((pStack[0]->pMem < &Mem[0]) ||
      (pStack[0]->pMem > pStack[1]->pMem + pStack[1]->MemSize)) rc=4;
  for ( int i=1; i<StackNum; i++ ) {
    if ((pStack[i]->pMem < pStack[i-1]->pMem + pStack[i-1]->MemSize) ||
      (pStack[i]->pMem > pStack[i+1]->pMem + + pStack[1]->MemSize)) rc=4;
  }
  if ( rc != 0 ) {
    printf("rc=%d\n",rc);
    res += 4;
  }
  return res;
}
            /*---------------------------------------------*/
```

```
// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// Copyright (c) Гергель В.П. 01.08.2000
//
// Динамические структуры данных - система N стеков
// наследование - распределение памяти по гипотезе
//   "сохранение локальных тенденций роста"

#ifndef __MULTISTACK1_H
#define __MULTISTACK1_H

#include "multist.h"

class TSuperMultiStack : public TMultiStack  {
  protected:
    int    PrevCount[StackNum];  // размер стеков при пред. перепаковке
    double StackRate[StackNum];  // показатели роста стеков
    void   SetStackRate     ( double StackRate[] ); // оценка показателей
    void   SetStackLocation ( TElem *pStackMem[] ); // оценка pStackMem
  public:
    TSuperMultiStack ();
};
#endif
```

```
// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// Copyright (c) Гергель В.П. 01.08.2000
//
// Динамические структуры данных - система N стеков
// наследование - распределение памяти по гипотезе
//   "сохранение локальных тенденций роста"

#include <stdio.h>
#include <conio.h>
#include <string.h>
#include "multist1.h"

TSuperMultiStack :: TSuperMultiStack () {
  for ( int i=0; i<StackNum; i++ ) {
    PrevCount[i] = 0; StackRate[i] = 0;
  }
}          /*--------------------------------------------*/
void TSuperMultiStack :: SetStackRate ( double StackRate[] ) {
// оценка показателей роста стеков
  double SumRate = 0.0;
  for ( int i=0; i<StackNum; i++ ) {
    if ( pStack[i]->DataCount < PrevCount[i] ) StackRate[i] = 0;
    else StackRate[i] = pStack[i]->DataCount - PrevCount[i];
    SumRate += StackRate[i];
  }
  if ( SumRate > 0 )
    for ( int i=0; i<StackNum; i++ ) StackRate[i] /= SumRate;
}          /*--------------------------------------------*/
void TSuperMultiStack :: SetStackLocation ( TElem *pStackMem[] ) {
// оценка новых значений для базовых адресов памяти стеков
// стратегия "сохранение локальных тенденций роста"
  SetStackRate(StackRate);
  pStackMem[0] = &Mem[0];
  for ( int i=1; i<StackNum; i++ )
    pStackMem[i] = pStackMem[i-1] + pStack[i-1]->DataCount
                         + int(StackRate[i-1]*FreeMemSize);
  pStackMem[StackNum] = pStack[StackNum]->pMem;
  for ( int i=0; i<StackNum; i++ )
    PrevCount[i] = pStack[i]->DataCount;
}
```

```
// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// Copyright (c) Гергель В.П. 01.08.2000
//
// Динамические структуры данных - система N стеков
//   распределение памяти по комбинированной схеме

#ifndef __MULTISTACK2_H
#define __MULTISTACK2_H

#include "multist1.h"

class TComplexMultiStack : public TSuperMultiStack  {
  protected:
    double MemQuota; // доля памяти, распределяемая по тенденции роста
  public:
    TComplexMultiStack ()              { MemQuota = 0.0;  }
    double GetMemQuota ( void ) const { return MemQuota; }
    void SetMemQuota ( double val )    { MemQuota = val;  }
    void SetStackLocation ( TElem *pStackMem[] ); // оценка pStackMem
};
#endif
```

```
// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// Copyright (c) Гергель В.П. 01.08.2000
//
// Динамические структуры данных - система N стеков
//   распределение памяти по комбинированной схеме

#include <stdio.h>
#include <conio.h>
#include <string.h>
#include "multist2.h"

void TComplexMultiStack :: SetStackLocation ( TElem *pStackMem[] ) {
// оценка новых значений для базовых адресов памяти стеков SKIP_ON
// смешанная стратегия
  SetStackRate(StackRate);
  pStackMem[0] = &Mem[0];
  for ( int i=1; i < StackNum; i++ )
    pStackMem[i] = pStackMem[i-1] + pStack[i-1]->DataCount
              + int( MemQuota * FreeMemSize / StackNum )          // поровну
              + int( (1-MemQuota) * StackRate[i-1]*FreeMemSize ); // темп роста
  pStackMem[StackNum] = pStack[StackNum]->pMem;
  for ( int i=0; i < StackNum; i++ )
    PrevCount[i] = pStack[i]->DataCount;                  // SKIP_OFF
}
```

```
// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// Copyright (c) Гергель В.П. 01.08.2000
//
// Динамические структуры данных - система N стеков
//   адаптивная схема подбора параметров

#ifndef __MULTISTACK3_H
#define __MULTISTACK3_H

#include "multist2.h"

class TAdaptMultiStack : public TComplexMultiStack  {
  protected:
    double QuotaStep; // шаг изменения доли памяти
    long   TimeStep;  // интервал времени для подсчета перепаковок
    long   PrevTime;  // время предыдущего шага адптации
    int    PrevCount; // к-во перепаковок на предыдущем шаге
    int    PrevInc;   // разность к-ва перепаковок на предыдущем шаге
  public:
    TAdaptMultiStack () {
      QuotaStep = 0.1; TimeStep = 2;
      PrevTime = PrevCount = PrevInc = 0;
    }
    void SetQuotaStep ( double step ) { QuotaStep = step; }
    void SetTimeStep  ( int    step ) { TimeStep  = step; }
    void SetStackLocation ( TElem *pStackMem[] ); // оценка pStackMem
};
#endif

// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// Copyright (c) Гергель В.П. 01.08.2000
//
// Динамические структуры данных - система N стеков
//   адаптивная схема подбора параметров

#include <time.h>
#include "multist3.h"

void TAdaptMultiStack :: SetStackLocation ( TElem *pStackMem[] ) {
// вставка схемы адаптации
  long NewTime;
  int  NewInc;
  TComplexMultiStack :: SetStackLocation ( pStackMem );
  NewTime = time( &NewTime );
  if ( NewTime > PrevTime + TimeStep ) { // адаптация
    NewInc = RelocationCount - PrevCount;
    if ( NewInc > PrevInc ) {
      MemQuota -= QuotaStep; if ( MemQuota < 0 ) MemQuota = 0.0;
    }
    else if ( NewInc < PrevInc ) {
      MemQuota += QuotaStep; if ( MemQuota > 1 ) MemQuota = 1.0;
    }
    PrevTime  = NewTime;
    PrevInc   = NewInc;
    PrevCount = RelocationCount;
  }
}
```

```
// Copyright (c) Гергель В.П. 01.08.2000
//
// Динамические структуры данных - система N стеков - тест

#include <dos.h>
#include <stdlib.h>
#include <conio.h>
#include <iostream.h>
#include "dataroot.h"
#include "datstack.h"
#include "datqueue.h"
#include "multroot.h"
#include "multist.h"

int main(int argc, char* argv[]) {
  TMultiStack mst;
  mst.Put(0,10);
  int ms=10, ns, code, temp, val=0;
  clrscr();
  cout << "Тестирование системы N стеков" << endl;
  while ( 1 ) { val++;
    code = random(4);     // операция
    ns   = random(ms);    // номер стека
    if ( code<3 ) mst.Put(ns,val);
    else temp=mst.Get(ns);
    if ( val % 100 == 0 ) {
      mst.Paint(2,2,79);
      gotoxy(1,3);
//      mst.IsValid();
      cout << "Циклов - " << val << ", Перепаковок - "
           << mst.GetRelocationCount() << endl;
    }
    if ( kbhit() ) break;
  }
  cout << "Циклов - " << val << ", Перепаковок - "
       << mst.GetRelocationCount() << endl;
  getch();
  return 0;
}
```

Нижегородский Государственный Университет им. Н.И. Лобачевского

Общий курс:

Методы программирования - 2

Тема 5:

Структуры хранения с использованием указателей (списки)

Гергель В.П., профессор
кафедра МО ЭВМ ВМК

Содержание

1.6. Структуры хранения с использованием указателей

1. Представление отношений следования с использованием указателей. Понятие линейного списка...

- ☑ Необходимость перепаковки для обеспечения динамического распределения памяти возникает в силу принятого способа реализации отношения следования - следующий элемент структуры располагается в следующем элементе памяти (с адресом, большим на 1)
- ☑ Устранение перепаковки возможно только при кардинальном изменении способа реализации основных отношений – необходимо допустить размещение следующих элементов структуры в произвольных элементах памяти (там, где имеется свободные области памяти)
- Возможность такого подхода может быть обеспечена запоминанием для каждого текущего элемента структуры адреса памяти, где хранится следующий элемент
- Интерпретация содержимого элемента памяти (значение или адрес следующего элемента) в самом простом варианте может быть обеспечена фиксированным форматом используемых участков памяти

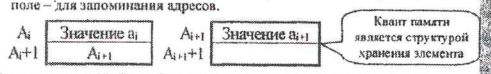
1.6. Структуры хранения с использованием указателей

1. Представление отношений следования с использованием указателей. Понятие линейного списка...

Определение 1.16. Под *квантом памяти* понимается последовательность элементов памяти с последовательно-возрастающими адресами. *Именем* (адресом) этой группы считается адрес первого слова кванта. Элементы кванта называются *полями*.

Определение 1.17. В общем случае, набор элементов памяти, связанных с одним именем, называют *звеном*.

☑ Далее будут использоваться двухэлементные звенья памяти, в которых первое поле будет использоваться для хранения значений, а второе поле – для запоминания адресов.

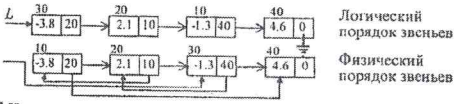


Определение 1.18. Способ задания отношения следования, в котором фиксация месторасположения следующего элемента производится путем запоминания соответствующего адреса памяти, называется *сцеплением* (пары, хранящие a_i и a_{i+1}, сцеплены адресными указателями).

1.6. Структуры хранения с использованием указателей

1. Представление отношений следования с использованием указателей. Понятие линейного списка

- ☑ Для изображения структуры хранения с использованием сцепления звенья памяти рисуются в виде прямоугольников, а сцепление звеньев показывается в виде стрелок.



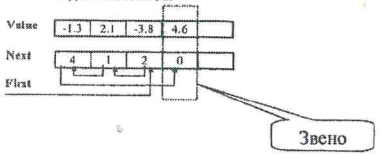
- ☑ Индикация последнего звена в списке обычно производится записью в поле адреса некоторого *барьера* – фиктивного (неадресного) значения (как правило 0 или -1).
- ☑ Для доступа к звеньям списка должен быть известен адрес первого звена списка. Указатель, в котором этот адрес запоминается, называется *переменной связи*.

Определение 1.19. Структура хранения данного типа (звенья, сцепление, барьер, переменная связи) называется *линейным* или *односвязным списком*.

1.6. Структуры хранения с использованием указателей

2. Реализация списков на языке высокого уровня...

Подход 1. Для имитации звеньев могут быть использованы два массива, один из которых используется для хранения значений, другой – для хранения индексов следующих элементов. В этом случае, звено есть элементы массивов с одинаковым индексом, адрес (имя) звена – индекс массивов.



2. Реализация списков на языке высокого уровня…

Подход 2. С использованием ООП звено может быть представлено в виде объекта. Образ памяти, выделенной для хранения структур данных, в этом случае будет представлять массив звеньев-объектов.

```
class TLink {
  public:
    int  Value; // значение
    int  Next;  // индекс следующего звена
  protected:
    TLink();
};
TLink Mem[MemLimit];
```


2. Реализация списков на языке высокого уровня

Практическая работа 6: Реализация стека

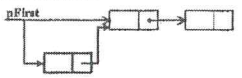
2. Сравнение непрерывной и списковой структур хранения

	Непрерывная память	Списки
1	Перепаковка для динамического распределения памяти	Динамическое распределение памяти эффективно реализуется при помощи списка свободных звеньев
2	В структуре хранения хранятся только данные	В структуре хранения хранятся данные и указатели
3	К элементам структуры данных обеспечивается прямой доступ	К элементам структуры данных обеспечивается последовательный доступ

3. Реализация списков с использованием динамически распределяемой памяти…

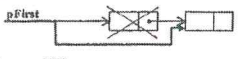
☑ Среда выполнения обеспечивает *динамически-распределяемую область памяти*:
- звено – ▷
- выделение звена – pTemp = new TDatLink()
- освобождение звена – delete pTemp

Вставка в стек




```
PTDatLink pTemp;
pTemp = new TDatLink();
pTemp->SetDatValue(Val);
pTemp->SetNextLink(pFirst);
pFirst = pTemp;
```

Выборка из стека



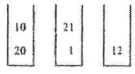

```
PTDatLink pTemp = pFirst;
Val = pFirst->GetDatValue();
pFirst = pFirst->GetNextLink();
delete pTemp;
```

Стек ▷ Пример: программа, приложение

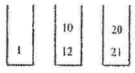
4. Пример использования стеков: поразрядная сортировка…

Пример 1.9 Упорядочение данных методом поразрядной сортировки (на примере набора значений 20 12 1 21 10)

- Разложить значения по стекам (номер стека - младшая цифра)



- Собрать значения из стеков (от старшего стека) 12 21 1 10 20



- Разложить значения по стекам (номер стека - старшая цифра)
- Собрать значения из стеков (от младшего стека) 1 10 12 20 21

Пример: программа, приложение

4. Пример использования стеков: преобразование арифметических выражений в польскую форму записи …

Пример 1.10. Преобразование выражений из инфиксной формы в польскую запись

Формат записи выражения
- Выражение синтаксически правильно (без ошибок)
- Допускаются только однобуквенные идентификаторы для операндов
- В записи выражения нет пробелов; выражение заканчивается знаком '='

A+(B-C)*D-F/(G+H)=

1.6. Структуры хранения с использованием указателей

4. Пример использования стеков: преобразование арифметических выражений в польскую форму записи …

Пример 1.10. Преобразование выражений из инфиксной формы в польскую запись

Алгоритм

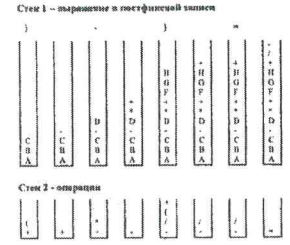
1. Для операций вводится приоритет
   '*' '/' (3), '+' '-' (2), '(' (1), '=' (0)
2. Для хранения данных используется 2 стека (1 – для результата, 2 – для операций)
3. Исходное выражение просматривается слева направо
4. Операнды по мере их появления помещаются в стек 1
5. Символы операций и левые скобки помещаются в стек 2
6. При появлении правой скобки последовательно изымаются элементы из стека 2 и переносятся в стек 1. Данные действия продолжаются либо до опустошения стека 2 либо до попадания в стеке 2 на левую скобку
7. Если текущая операция, выделенная при обходе выражения, имеет меньший (более низкий) приоритет, чем операция на вершине стека 2, то такие операции из стека 2 переписываются в стек 1

1.6. Структуры хранения с использованием указателей

4. Пример использования стеков: преобразование арифметических выражений в польскую форму записи …

Пример 1.10. Преобразование выражений из инфиксной формы в польскую запись

Пусть выражение имеет вид A+(B-C)*D-F/(G+H)=



программа, приложение

1.6. Структуры хранения с использованием указателей

5. Обеспечения списковой структуры хранения

Практическая работа 7: Разработка общего представления линейного списка

1.6. Структуры хранения с использованием указателей

6. Стандартная библиотека шаблонов алгоритмического языка C++…

☑ В стандарте языка C++ предусматривается наличие в среде программирования *стандартной библиотеки шаблонов* (*Standard Template Library, STL*)

Основные понятия библиотеки STL

I. Библиотека включает в свой состав большое количество ***контейнеров***, представляющих собой структуры данных, в которых могут храниться объекты. В числе имеющихся контейнеров

- vector<T> - вектор переменного размера,
- list<T> - двусвязный список,
- queue<T> - очередь,
- stack<T> - стек,
- deque<T> - дек,
- priority_queue<T> - приоритетная очередь,
- set<T> - множество,
- multiset<T> - множество с повторением элементов,
- map<key,val> - ассоциативный массив (таблица),
- multimap<key,val> - ассоциативный массив с повторением ключей

1.6. Структуры хранения с использованием указателей

6. Стандартная библиотека шаблонов алгоритмического языка C++…

II. Для быстрого и эффективного построения вычислительных процедур, библиотека обеспечивает ***итераторы*** для всех видов контейнеров, которые представляют унифицированный механизм последовательного доступа к элементам контейнеров.

Общая схема:

- <класс-контейнер>::iterator Iter; - объявление итератора,
- Iter = <объект-контейнер>.begin(); - установка на первый элемент,
- Iter != <объект-контейнер>.end(); - проверка на завершение,
- ++Iter – переход к следующему элементу

В зависимости от типа контейнера, итератор может обеспечивать прямой доступ, быть одно- или двух- направленным, предназначенным только для чтения или записи и др.

1.6. Структуры хранения с использованием указателей

6. Стандартная библиотека шаблонов алгоритмического языка C++…

III.Библиотека содержит для контейнеров большое количество реализованных ***обобщенных алгоритмов***. В числе таких алгоритмов:

- for_each() - вызвать функцию для каждого элемента,
- find() - найти первое вхождение элемента,
- find_if() - найти первое соответствие условию,
- count() - подсчитать число вхождений элемента,
- count_if() - подсчитать число соответствий условию,
- replace() - заменить элемент новым значением,
- copy() - скопировать элементы,
- unique_copy() - скопировать только различные элементы,
- sort() - отсортировать элементы,
- merge() - объединить отсортированные последовательности

и др.

1.6. Структуры хранения с использованием указателей

6. Стандартная библиотека шаблонов алгоритмического языка C++

IV. Покажем для примера использование контейнера списка.

Пример: программа, приложение

Заключение

- Реализация отношения следования при помощи сцепления (адресных указателей)
- Линейный список как структура хранения
- Реализация списков на языках высокого уровня
- Реализация списков с использованием динамически-распределяемой области памяти
- Использование списковых структур хранения на примере реализации стеков
- Разработка общего представления линейного списка для обеспечения списковой структуры хранения
- Общая характеристика стандартной библиотеки шаблонов

Вопросы для обсуждения

- Сравнение непрерывной и списковых структур хранения
- Разработка стандартных программ для работы со списками

Темы заданий для самостоятельной работы

- Реализация очередей с использованием списков
- Реализация двунаправленных списков

Следующая тема

- Структуры данных и конструирование математических моделей

```
// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// datlink.h Copyright (c) Гергель В.П. 09.08.2000
//
// Списки - класс для звена списка

#ifndef __DATLINK_H
#define __DATLINK_H

#include <iostream.h>

class TDatLink;
typedef TDatLink *PTDatLink;
typedef int       TData;     // тип значений в СД

class TDatLink {
  protected:
    TData      Value;  // значение
    PTDatLink  pNext;  // указатель на следующее звено
  public:
    TDatLink ( TData Val = 0 ) {
      Value = Val; pNext = NULL;
    }
    void       SetDatValue ( TData Val )          { Value  = Val;   }
    TData      GetDatValue ()                     { return  Value;  }
    void       SetNextLink ( PTDatLink  pLink   ) { pNext  = pLink; }
    PTDatLink  GetNextLink () { return  pNext;  }
    friend class TList;
};
#endif

// end of datlink.h

// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// Copyright (c) Гергель В.П. 06.08.2000
//
// Динамические структуры данных - стек (списки)

#ifndef __DATLIST_H
#define __DATLIST_H

#define DataEmpty  -101  // СД пуста
#define DataFull   -102  // СД переполнена
#define DataNoMem  -103  // нет памяти

#include "datacom.h"
#include "datlink.h"

class TListStack : public TDataCom {
  protected:
    PTDatLink pFirst; // указатель на первое звено списка
  public:
    TListStack () { pFirst = NULL; }
    int IsFull  (void) const;        // контроль переполнения СД
    int IsEmpty (void) const;        // контроль переполнения СД
    virtual void Put ( TData Val );  // добавить значение в стек
    virtual TData Get(void);         // извлечь значение из стека (#Л1)
    // служебные методы
    virtual void Print();                    // печать значений
//   virtual void Paint(int y,int x1,int x2) {} // показать рисунок структуры
//    virtual int  IsValid()       { return 0; } // тестирование структуры
//    virtual void CopyToQueue(TQueue *pQ)    {} // копировать в очередь
};
#endif
```

```
// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// Copyright (c) Гергель В.П. 27.07.2000
//
// Динамические структуры данных - стек - версия 2.2

#include "datlist.h"

int TListStack :: IsFull(void) const { // контроль пустоты СД
  PTDatLink pTemp = new TDatLink();
  int res = (pTemp==NULL) ? 1 : 0;
  delete pTemp;
  return res;
}
            /*-------------------------------------------*/

int TListStack :: IsEmpty(void) const { // контроль переполнения СД
  return pFirst == NULL;
}
            /*-------------------------------------------*/

void TListStack :: Put ( TData Val ) { // добавить значение в стек
  if ( IsFull() ) SetRetCode ( DataFull );
  else {
    PTDatLink pTemp = new TDatLink();
    pTemp->SetDatValue(Val);
    pTemp->SetNextLink(pFirst);
    pFirst = pTemp;
    SetRetCode ( DataOK );
  }
}
            /*-------------------------------------------*/

TData TListStack :: Get ( void ) { // извлечь значение из стека
  TData Val;                              // SKIP_ON
  if ( IsEmpty() ) SetRetCode ( DataEmpty );
  else {
    PTDatLink pTemp = pFirst;
    Val = pFirst->GetDatValue();
    pFirst = pFirst->GetNextLink();
    delete pTemp;
    SetRetCode ( DataOK );
  }
  return Val;                             // SKIP_OFF
}
            /*-------------------------------------------*/

void TListStack :: Print(void) { // печать стека
  PTDatLink pTemp = pFirst;
  cout << "Печать стека (от вершины)" << endl;
  while ( pTemp != NULL ) {
    cout << pTemp->GetDatValue() << " ";
    pTemp = pTemp->GetNextLink();
  }
  cout << endl;
}
```

```
// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// Copyright (c) Гергель В.П. 05.08.2000
//
// Динамические структуры данных - тестирование системы N стеков (списки)

#include <dos.h>
#include <stdlib.h>
#include <conio.h>
#include <iostream.h>
#include "datlist.h"

void main() {
  TListStack mst;
  int code, temp, val=0;
  clrscr();
  cout << "Тестирование системы поддержки стеков (списки)" << endl;
  cout << "Нажмите любую клавишу" << endl;
  while ( 1 ) { val++;
    code = random(4);     // операция
    if ( code<2 ) mst.Put(val);
    else temp=mst.Get();
    if ( kbhit() ) break;
  }
  getch();
  cout << "Печать стека" << endl;
  mst.Print();
  cout << "Нажмите любую клавишу" << endl;
  getch();
  while ( !mst.IsEmpty() ) temp=mst.Get();
}
```

```
// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// Copyright (c) Гергель В.П. 05.08.2000
//
// Динамические структуры данных - тестирование системы N стеков (списки)

#include <dos.h>
#include <stdlib.h>
#include <conio.h>
#include <iostream.h>
#include "datlist.cpp"

#define N  15
#define NS 10 // количество стеков

void main() {
  TListStack mst[NS];
  int i, k, val[N];
  clrscr();
  cout << "Поразрядная сортировка - пример на использование стеков" << endl;
  // генерация данных
  cout << endl;
  for ( i=0; i<N; i++ ) val[i] = random(100);
  cout << "Данные до сортировки" << endl;
  for ( i=0; i<N; i++ ) cout << val[i] << " ";
  cout << endl;

  // раскладка по стекам (номер стека - младшая цифра)
  cout << endl;
  for ( i=0; i<N; i++ ) mst[val[i]%10].Put(val[i]);

  // сборка значений из стеков (от старшего стека)
  for ( i=NS-1, k=0; i>-1; i-- )
    while ( ! mst[i].IsEmpty() ) val[k++] = mst[i].Get();
  cout << "Данные после первой раскладки" << endl;
  for ( i=0; i<N; i++ ) cout << val[i] << " ";
  cout << endl;

  // раскладка по стекам (номер стека - старшая цифра)
  cout << endl;
  for ( i=0; i<N; i++ ) mst[val[i]/10].Put(val[i]);

  // сборка значений из стеков (от младшего стека)
  for ( i=0, k=0; i<NS; i++ )
    while ( ! mst[i].IsEmpty() ) val[k++] = mst[i].Get();
  cout << "Данные после второй раскладки (итог)" << endl;
  for ( i=0; i<N; i++ ) cout << val[i] << " ";
  cout << endl;

  getch();
}
```

```
// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// Copyright (c) Гергель В.П. 25.07.2002
//
// Перевод арифметических выражений из инфиксной формы в польскую запись

#include <dos.h>
#include <stdlib.h>
#include <conio.h>
#include <iostream.h>
#include "datlist.h"

class TInfixToPolish {
  protected:
    int GetOperationPrt(char op); // получить приоритет операции
    int IsOperation(char op);     // проверка на знак операции
  public:
    char * ConvertToPolish(char * Exp, int len); // проебразование к польской
записи
};

int TInfixToPolish :: GetOperationPrt(char op) { // получить приоритет операции
  int Prt;
  switch ( op ) {
     case '*':
     case '/': Prt = 3; break;
     case '+':
     case '-': Prt = 2; break;
     case '(': Prt = 1; break;
     case '=': Prt = 0; break;
     default:  Prt = -1;
  }
  return Prt;
}

int TInfixToPolish :: IsOperation(char op) { // проверка на знак операции
  if ( op=='+' || op=='-' || op=='*' || op=='/' || op=='=' ) return 1;
  else return 0;
}

// проебразование выражения от инфиксной формы к польской записи
// выражение правильное, без пробелов, заканчивается знаком '='
char * TInfixToPolish :: ConvertToPolish(char * InfixExp, int len) {
  char ch, t, *PolishExp = new char[strlen(InfixExp)+1];
  int  pos = 0; // индекс текущего символа в выражении
  TListStack PolishStack, OperationStack;
  bool key;
  do {
    ch  = InfixExp[pos++];
    if ( isalpha(ch) ) PolishStack.Put(ch); // операнд
    else if ( ch == '(' ) OperationStack.Put(ch);
    else if ( ch == ')' ) {
      while (1) { // перепись операций из стека операций до откр. скобки
        t = OperationStack.Get();
        if ( t == '(' ) break;
        PolishStack.Put(t);
      }
    } else if ( IsOperation(ch) ) { // операции с меньшим приоритетом
        while (! OperationStack.IsEmpty() ) {  // в стек результатов
          t = OperationStack.Get();
          if ( GetOperationPrt(ch) <= GetOperationPrt(t) ) PolishStack.Put(t);
          else { OperationStack.Put(t); break; }
        }
        OperationStack.Put(ch);
```

```
    } // '=' в стеке операций
  } while ( (ch != '=') && (pos < len) );
  pos = 0;  // длина выражения в польской записи
  for ( int i=0; i < len; i++ )
    if ( (InfixExp[i] != '(' ) && (InfixExp[i] != ')' ) ) pos++;
  PolishExp[pos]   = '\0';
  PolishExp[--pos] = '=';
  // извлечение выражения из стека - порядок обратный
  while ( ! PolishStack.IsEmpty() ) PolishExp[--pos] = PolishStack.Get();
  return PolishExp;
}

int main() {
  TInfixToPolish ExpConvertor;
  char Expression[80], *PolishExpression;
//  cout << "Перевод арифм. выражения из инфиксной в постфиксную запись" <<
endl;
//  cout << "Введите выражение" << endl;
  cout << "Converting the expression form infix form to polish" << endl;
  cout << "Input the expression (last symbol has to be '='" << endl;
  cin >> Expression;
  PolishExpression = xpConvertor.ConvertToPolish(Expression,strlen(Expression));
  cout << "Выражение в инфиксной записи - " << Expression << endl;
  cout << "Выражение в обратной польской записи - " << PolishExpression << endl;
  delete PolishExpression;
  cout << "Нажмите любую клавишу..." << endl;
  getch();
}
```

```
// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// Copyright (c) Гергель В.П. 25.07.2002
//
// Пример использования стандартной библиотеки шаблонов
//
#include <iostream.h>
#include <list>
#include <string>
#include <conio.h>

using namespace std;

// Переименование некоторых типов для облегчения
// последующей работы при создании списка и его
// итератора.
//
typedef list<string>           strList;
typedef list<string>::iterator strIter;

int main(int argc, char* argv[]) {

  strList myList;
//  cout << "      Тестирование списка библиотеки STL" << endl;
  cout << "      Test for list container from STL" << endl;
  cout << endl;
  // Добавление строк
  //
  myList.insert(myList.end(),"first");
  myList.insert(myList.end(),"second");
  myList.insert(myList.end(),"third");
  myList.insert(myList.end(),"fourth");
  myList.insert(myList.end(),"fifth");
  myList.push_front("Head"); // добавить в начало
  myList.push_back("Tail");  // добавить в конец

  // Вывод списка
  //
  cout << "   Печать списка" << endl;
  for (strIter iter=myList.begin(); iter != myList.end(); ++iter )
    cout << *iter << endl;
  cout << endl; // Добавление пустой строки

  // Убираем один из элементов
  //
  myList.erase(find(myList.begin(), myList.end(), "third"));

  // Сейчас показываем в обратном порядке
  //
  cout << "   Печать списка в обратном порядке" << endl;
  strIter iter = myList.end();
  --iter;
  for (int ix = myList.size(); ix > 0; --iter, --ix)
    cout << *iter << endl;
  cout << endl;
  cout << "   Нажмите любую клавишу" << endl;
  getch();
  return 0;

}
```

Нижегородский Государственный Университет им. Н.И. Лобачевского

Общий курс:

Методы программирования - 2

Практическая работа 6:

Структура хранения нескольких стеков с использованием списков на языке высокого уровня

Гергель В.П., профессор
кафедра МО ЭВМ ВМК

Содержание

1. Структура звена списка

Звено списка представляется в виде объекта класса TLink

```
class TLink {
  public:
    int  Value; // значение
    int  Next;  // индекс следующего звена
  protected:
    TLink();
};
```


2. Структура памяти

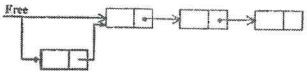
Образ памяти, выделенной для хранения стека, определяется в виде массива звеньев-объектов

```
TLink Mem[MemLimit];
```


3. Организация списка свободных звеньев

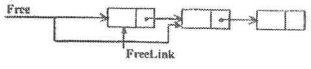
☑ Все свободные звенья объединяются в один список свободных звеньев. Звенья этого списка используются при необходимости свободной памяти, в этот список звенья должны возвращаться после освобождения.

Вставка звена в список свободных звеньев



Новое звено включается в начало списка свободных

Выборка звена из списка свободных звеньев

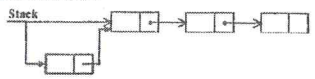


Для выделения используется первое звено списка свободных

4. Структура хранения стека

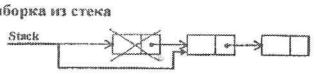
☑ Структура хранения стека – линейный список (начало списка – вершина стека).

Вставка в стек



Звено для нового значения берется в списке свободных

Выборка из стека



Исключаемое звено из стека должно оказаться в списке свободных

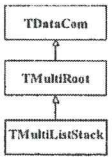
☑ Схема работы со стеком и со списком свободных совпадают

↳ Список свободных звеньев есть стек.

Структуры хранения стека с использованием списков

5. Реализация

Схема наследования



Контрольный пример: программа, приложение

Заключение

- Представление звеньев списка в виде объектов
- Представление памяти виде массива звеньев
- Организация списка свободных звеньев
- Структура хранения стека в виде списка

Вопросы для обсуждения

- Сравнение способов организации динамического распределения памяти (перепаковка, списки)
- Проблема статического определения максимального размера памяти

Темы заданий для самостоятельной работы

- Разработка тестов для проверки правильности работы программ (контроль связности списка, контроль "потерянных" звеньев,…)
- Разработка структуры хранения очереди с использованием списков

Следующая тема

- Реализация списков с использованием динамически-распределяемой области памяти

```
// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// Copyright (c) Гергель В.П. 05.08.2000
//
// Динамические структуры данных - система N стеков (списки)

#ifndef __MULTLIST_H
#define __MULTLIST_H

#include "dataroot.h"
#include "multroot.h"

class TMultiListStack : public TMultiRoot {
  protected:
    int NextLink[MemLimit];          // индекс следующего звена
    int StackInd[StackNum];         // индекс вершин стеков
    int FirstFreeLink;              // индекс первого свободного звена
  public:
    TMultiListStack ();
    int  IsEmpty ( int ns ) const; // контроль пустоты СД
    int  IsFull  ( int ns ) const; // контроль переполнения СД
    virtual void  Put ( int ns, const TData &Val ); // положить в стек
    virtual TData Get ( int ns );  // взять из стека с удалением
// служебные методы
    virtual void Print();                // печать значений стеков
    virtual int  IsValid() { return 0; } // тестирование структуры
};
#endif

// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// Copyright (c) Гергель В.П. 05.08.2000
//
// Динамические структуры данных - система N стеков (списки)

#include <stdio.h>
#include "multlist.h"

TMultiListStack :: TMultiListStack () {
  for ( int i=0; i<MemLimit; i++ ) NextLink[i] = i+1;
  NextLink[MemLimit-1] =-1;
  FirstFreeLink       = 0;
  for ( int i=0; i<StackNum; i++ )    StackInd[i] = -1;
}           /*-------------------------------------------*/

int TMultiListStack :: IsEmpty ( int ns ) const { // контроль пустоты СД
  return StackInd[ns] < 0;
}           /*-------------------------------------------*/

int TMultiListStack :: IsFull ( int ns ) const { // контроль переполнения СД
  return FirstFreeLink < 0;
}           /*-------------------------------------------*/

void TMultiListStack :: Put ( int ns, const TData &Val ) { // положить в стек
  if ( IsFull(ns) ) SetRetCode ( DataFull );
  else {
    int k = FirstFreeLink;
    FirstFreeLink = NextLink[k];
    Mem[k]        = Val;
    NextLink[k]   = StackInd[ns];
    StackInd[ns]  = k;
    SetRetCode ( DataOK );
  }
} // Put
```

```
TData TMultiListStack :: Get ( int ns ) { // взять из стека с удалением
  TData temp = -1;
  if ( IsEmpty(ns) ) SetRetCode ( DataEmpty );
  else {
    int k = StackInd[ns];
    temp  = Mem[k];
    StackInd[ns]  = NextLink[k];
    NextLink[k]   = FirstFreeLink;
    FirstFreeLink = k;
    SetRetCode ( DataOK );
  }
  return temp;
} // Get
          /*-----------------------------------------------*/

void TMultiListStack :: Print() { // печать значений стеков
  int pind, ind, k;
  for ( int ns=0; ns<StackNum; ns++ ) {
    printf("ns=%d -> ",ns);
    pind = -1; ind  = StackInd[ns];
    while ( ind > -1 ) { // оборачивание указателей
      k    = NextLink[ind];
      NextLink[ind] = pind;
      pind = ind; ind  = k;
    }
    ind = pind; pind = -1;
    while ( ind > -1 ) { // восстановление указателей и печать
      printf("%d ",Mem[ind]);
      k    = NextLink[ind];
      NextLink[ind] = pind;
      pind = ind; ind  = k;
    }
    printf("\n");
  }
}

// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// Copyright (c) Гергель В.П. 05.08.2000
//
// Динамические структуры данных - тестирование системы N стеков (списки)

#include <dos.h>
#include <stdlib.h>
#include <conio.h>
#include <iostream.h>
//#include "dataroot.h"
#include "multroot.h"
#include "multlist.h"

void main() {
  TMultiListStack mst;
  int ms=10, ns, code, temp, val=0;
  clrscr();
  cout << "Тестирование системы N стеков (списки)" << endl;
  cout << "Нажмите любую клавишу" << endl;
  while ( 1 ) { val++;
    code = random(4);     // операция
    ns   = random(ms);    // номер стека
    if ( code<3 ) mst.Put(ns,val); else temp=mst.Get(ns);
    if ( kbhit() ) break;
  }
  cout << "Печать стеков" << endl;
  mst.Print();
}
```

Нижегородский Государственный Университет им. Н.И. Лобачевского

Общий курс:

Методы программирования - 2

Практическая работа 7:

Разработка общего представления линейного списка

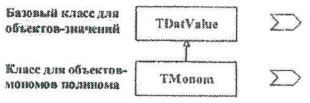
Гергель В.П., профессор
кафедра МО ЭВМ ВМК

Содержание

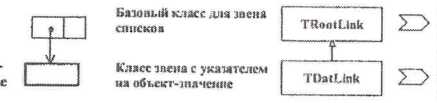
Разработка общего представления линейного списка

1. Организация хранения в списках значений разного типа

- Представим значений в виде объектов классов, являющиеся производными из одного общего базового класса

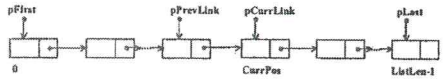


- В поле значения звена списка будем размещать указатель на объект значение



Разработка общего представления линейного списка

2. Проектирование структуры списка



- pFirst – указатель на первое звено списка
- pLast – указатель на последнее звено списка
- pCurrLink – указатель на текущее звено списка
- pPrevLink – указатель на звено, предшествующее текущему
- CurrPos – номер текущего звена
- ListLen – количество звеньев в списке

☑ Для повышения общности схемы реализации будем использовать вместо величины NULL константу pStop для фиксации ситуаций, в которых указатель не содержит адрес какого-либо звена списка

Разработка общего представления линейного списка

3. Операции для работы со списком…

- Информационные методы
  - IsEmpty – проверить, не является ли список пустым
  - GetListLength – получить количество звеньев списка
- Методы доступа к значениям в списке
  - GetDatValue – получить указатель на значение из звена списка (обращение возможно только в первому (FIRST), текущему (CURRENT) или последнему звеньям списка (LAST); желаемый вариант доступа задается через параметр метода)

Разработка общего представления линейного списка

3. Операции для работы со списком…

- Методы навигации по списку (*итератор*)
  - Reset – установить текущую позицию на первое звено
  - GoNext – переместить текущую позицию на звено вправо
  - IsListEnded – проверка завершения списка (под ситуацией завершения списка понимается состояние после применения GoNext для текущей позиции, установленной на последнем звене списка, т.е. когда pPrevLink=pLast, pCurrLink=pStop)
  - GetCurrentPos – получить номер текущего звена
  - SetCurrentPos – установить текущую позицию на звено с заданным номером (прямой доступ к звеньям !?)

Разработка общего представления линейного списка

3. Операции для работы со списком…

- Вставка звеньев
  - InsFirst – вставить звено перед первым звеном списка
  - InsLast – вставить звено после последнего звена
  - InsCurrent – вставить звено перед текущим звеном списка
- ☑ При выполнении операций вставки звеньев следует учитывать, что список может быть пустым
- ☑ После выполнения вставки необходимо обеспечить корректность значений указателей первого, текущего и последнего звеньев списка
- ☑ При корректировке указателей следует учитывать возможность различного положения текущей позиции списка
  - текущая позиция является первым звеном (pCurrLink=pFirst),
  - текущая позиция является вторым звеном (pPrevLink=pFirst),
  - текущая позиция находится внутри списка,
  - текущая позиция является последним звеном (pCurrLink=pLast),
  - текущая позиция выходит за пределы списка (pPrevLink=pLast)

Разработка общего представления линейного списка

3. Операции для работы со списком

- Удаление звеньев
  - DelFirst – удалить первое звено списка
  - DelCurrent – удалить текущее звено
  - DelList – удалить список

Пример: программа, приложение

Разработка общего представления линейного списка

4. Обеспечение удобного интерфейса…

- Надежность преобразование типа
  - ☑ Указатель производного типа может автоматически приводиться к указателю на базовый тип
  - ☑ Для обратного приведения – от базового типа к производному – необходимо явное преобразования типа

```
PTDatValue pValue;
PTMonom pMonom;
pMonom = PTMonom(pValue);
```

⇒ такое преобразование типа не является безопасным

  - ☑ Преобразование и использованием информацией о типе времени исполнения

```
pMonom = dynamic_cast<PTMonom>(pValue);
```

⇒ преобразование типа выполняется только если тип находится выше по иерархии наследования от объекта, указываемого pVal (иначе pMonom=NULL)

Разработка общего представления линейного списка

4. Обеспечение удобного интерфейса…

- Владение (создание и удаление) значениями
  - ☑ Использование нескольких указателей на объект усложняет контроль за его временем жизни
  - ☑ Возможное решение – передача объектов *по значению*
    - при записи в список указателя на объект в списке запоминается указатель на копию объекта-значения,
    - при получении указателя из списка создается еще одна копия объекта-значения и как результат передается указатель на новую созданную копию

⇒ Возможность создания копий обеспечивает метод GetCopy

Разработка общего представления линейного списка

4. Обеспечение удобного интерфейса

- Обеспечение удобного API
  - ☑ Работа с объектами является более удобным по сравнению с использованием указателей

⇒ Все перечисленные моменты (безопасное преобразование типов, создание копий, использование объектов) могут быть учтены при помощи создания шаблона класса-переходника (*proxy*) TList к классу TDatList

Пример: программа

Заключение

- Организация хранения объектов-значений разных типов
- Проектирование структуры списка
- Операции для работы со списком
- Обеспечения удобного интерфейса (классы-переходники)

Вопросы для обсуждения

- Организация навигации по структуре данных (итераторы)
- Унификация работы с данными (классы-оболочки)

Темы заданий для самостоятельной работы

- Расширение набора операций со списком (поиск по значению или условию, совмещение итератора с операциями обработки,…)
- Разработка двунаправленного списка (наследование)

Следующая тема

- Общая характеристика стандартной библиотеки шаблонов)

```
// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// datvalue.h Copyright (c) Гергель В.П. 09.08.2000
//
// Списки - базовый (абстрактный) класс объектов-значений

#ifndef __DATVALUE_H
#define __DATVALUE_H

#include <iostream.h>
class TDatValue {
  public:
    virtual TDatValue * GetCopy() =0; // создание копии
   ~TDatValue() {}
};
typedef TDatValue *PTDatValue;
#endif

// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// tmonom.h Copyright (c) Гергель В.П. 09.08.2000
//
//  класс объектов-значений для мономов полинома

#ifndef __TMONOM_H
#define __TMONOM_H

#include <iostream.h>
#include "datvalue.h"
class TMonom : public TDatValue  {
  protected:
    int Coeff; // коэффициент монома
    int Index; // индекс (свертка степеней)
  public:
    TMonom ( int cval=1, int ival=0 ) { Coeff=cval; Index=ival; }
    virtual TDatValue * GetCopy(); // изготовить копию
    void SetCoeff(int cval) { Coeff=cval;   }
    int  GetCoeff(void)     { return Coeff; }
    void SetIndex(int ival) { Index=ival;   }
    int  GetIndex(void)     { return Index; }
    TMonom& operator=(const TMonom &tm) {
      Coeff=tm.Coeff; Index=tm.Index; return *this;
    }
    int   operator==(const TMonom &tm) {
      return (Coeff==tm.Coeff) && (Index==tm.Index);
    }
    int   operator< (const TMonom &tm) { return Index<tm.Index; }
    friend ostream& operator<<(ostream &os,TMonom &tm) {
      os << tm.Coeff << " " << tm.Index;  return os;
    }
    friend class TPolinom;
};
typedef TMonom *PTMonom;
#endif

// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// tmonom.cpp - Copyright (c) Гергель В.П. 09.08.2000
//
//   класс объектов-значений для мономов полинома

#include "tmonom.h"
TDatValue * TMonom :: GetCopy() { // изготовить копию
  TDatValue *temp = new TMonom(Coeff,Index); return temp;
}
```

```
// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// rootlink.h Copyright (c) Гергель В.П. 09.08.2000, 25.07.2002
//
// Базовый класс для звеньев

#ifndef __ROOTLINK_H
#define __ROOTLINK_H

#include <iostream.h>
#include "datvalue.h"

class TRootLink;
typedef TRootLink *PTRootLink;

class TRootLink {
  protected:
    PTRootLink  pNext;  // указатель на следующее звено
  public:
    TRootLink ( PTRootLink pN = NULL ) { pNext = pN; }
    PTRootLink  GetNextLink () { return  pNext; }
    void        SetNextLink ( PTRootLink  pLink ) { pNext  = pLink; }
    void        InsNextLink ( PTRootLink  pLink ) {
      PTRootLink p = pNext;  pNext  = pLink;
      if ( pLink != NULL ) pLink->pNext = p;
    }
    virtual void       SetDatValue ( PTDatValue pVal ) = 0;
    virtual PTDatValue GetDatValue ()                  = 0;

    friend class TDatList;
};
#endif

// end of rootlink.h

// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// datlink.h Copyright (c) Гергель В.П. 09.08.2000
//
// Списки - класс для звена списка

#ifndef __DATLINK_H
#define __DATLINK_H

#include <iostream.h>
#include "rootlink.h"

class TDatLink;
typedef TDatLink *PTDatLink;

class TDatLink : public TRootLink {
  protected:
    PTDatValue pValue;  // указатель на объект значения
  public:
    TDatLink ( PTDatValue pVal = NULL, PTRootLink pN = NULL ) : TRootLink(pN) {
      pValue = pVal;
    }
    void       SetDatValue ( PTDatValue pVal ) { pValue = pVal; }
    PTDatValue GetDatValue    () { return  pValue;           }
    PTDatLink  GetNextDatLink () { return  (PTDatLink)pNext; }
    friend class TDatList;
};
#endif
// end of datlink.h
```

```
// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// datlist.h - Copyright (c) Гергель В.П. 09.08.2000
//
// Списки

#ifndef __DATLIST_H
#define __DATLIST_H

#include "datacom.h"
#include "datlink.h"

#define ListOK         0  // ошибок нет
#define ListEmpty   -101  // список пуст
#define ListNoMem   -102  // нет памяти
#define ListNoPos   -103  // ошибочное положение
                          // текущего указателя
enum TLinkPos { FIRST, CURRENT, LAST };

class TDatList : public TDataCom {
  protected:
    PTDatLink pFirst;     // первое звено
    PTDatLink pLast;      // последнее звено
    PTDatLink pCurrLink;  // текущее звено
    PTDatLink pPrevLink;  // звено перед текущим
    PTDatLink pStop;      // значение указателя, означающего конец списка (=NULL)
    int CurrPos;          // номер текущего знена (нумерация от 0)
    int ListLen;          // количество звеньев в списке
  protected:  // методы
    PTDatLink GetLink ( PTDatValue pVal=NULL, PTDatLink pLink=NULL );
    void      DelLink ( PTDatLink pLink );   // удаление звена
  public:
    TDatList();
    ~TDatList() { DelList(); }
    // доступ
    PTDatValue GetDatValue ( TLinkPos mode = CURRENT ) const; // значение
    virtual int IsEmpty()  const { return pFirst==pStop; }    // список пуст ?
    int GetListLength()    const { return ListLen; }          // к-во звеньев
    // навигация
    int SetCurrentPos ( int pos );          // установить текущее звено
    int GetCurrentPos ( void ) const;       // получить номер текущего звена
    virtual int Reset ( void );             // установить на начало списка
    virtual int IsListEnded ( void ) const; // список завершен ?
    int GoNext ( void );                    // сдвиг вправо текущего звена
              // (=1 после применения GoNext для последнего звена списка)
    // вставка звеньев
    virtual void InsFirst  ( PTDatValue pVal=NULL ); // вставить перед первым
    virtual void InsLast   ( PTDatValue pVal=NULL ); // вставить последним      (#Л1)
    virtual void InsCurrent( PTDatValue pVal=NULL ); // вставить перед текущим (#П1)
    // удаление звеньев
    virtual void DelFirst  ( void );                 // удалить первое звено   (#Л2)
    virtual void DelCurrent( void );                 // удалить текущее звено  (#П2)
    virtual void DelList   ( void );                 // удалить весь список
};
typedef TDatList *PTDatList;
#endif
```

```
// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// datlist.cpp - Copyright (c) Гергель В.П. 09.08.2000
//
// Списки

#include "datlist.h"

TDatList :: TDatList() {
  pFirst = pLast = pStop = NULL; ListLen = 0;
  Reset();
}              /*-----------------------------------------------*/

PTDatLink TDatList :: GetLink ( PTDatValue pVal, PTDatLink pLink ) {
  PTDatLink temp = new TDatLink(pVal, pLink);  // выделение звена
  if ( temp == NULL ) SetRetCode(ListNoMem); else SetRetCode(ListOK);
  return temp;
}              /*-----------------------------------------------*/

void TDatList :: DelLink ( PTDatLink pLink ) { // удаление звена
  if ( pLink != NULL ) {
    if ( pLink->pValue != NULL ) delete pLink->pValue;
    delete pLink;
  }
  SetRetCode(ListOK);
}              /*-----------------------------------------------*/

// методы доступа

PTDatValue TDatList :: GetDatValue ( TLinkPos mode ) const { // значение
  PTDatLink temp;
  switch ( mode ) {
    case FIRST: temp = pFirst;    break;
    case LAST:  temp = pLast;     break;
    default:    temp = pCurrLink; break;
  }
  return (temp==NULL) ? NULL : temp->pValue;
}              /*-----------------------------------------------*/

// методы навигации

int TDatList :: SetCurrentPos ( int pos ) {    // установить текущее звено
  Reset();
  for ( int i=0; i<pos; i++, GoNext() )
  SetRetCode(ListOK); return RetCode;
}              /*-----------------------------------------------*/

int TDatList :: GetCurrentPos ( void ) const { // получить номер текущего звена
  return CurrPos;
}              /*-----------------------------------------------*/

int TDatList :: Reset ( void ) {               // установить на начало списка
  pPrevLink = pStop;
  if (IsEmpty()) { pCurrLink = pStop;  CurrPos =-1; SetRetCode(ListEmpty); }
  else           { pCurrLink = pFirst; CurrPos = 0; SetRetCode(ListOK);    }
  return RetCode;
}              /*-----------------------------------------------*/

int TDatList :: GoNext ( void ) {              // сдвиг вправо текущего звена
  if ( pCurrLink == pStop ) SetRetCode(ListNoPos);
  else {
    SetRetCode(ListOK);
    pPrevLink = pCurrLink; pCurrLink = pCurrLink->GetNextDatLink(); CurrPos++;
  }
  return RetCode;
}              /*-----------------------------------------------*/
```

```
int TDatList :: IsListEnded ( void ) const {    // список завершен ?
  // (=1 после применения GoNext для последнего звена списка)
  return pCurrLink == pStop;
}             /*-----------------------------------------------*/

// методы вставки звеньев

void TDatList :: InsFirst ( PTDatValue pVal ) { // вставить перед первым
  PTDatLink temp = GetLink(pVal,pFirst);
  if ( temp == NULL ) SetRetCode(ListNoMem);
  else {
    pFirst = temp; ListLen++;
    // проверка пустоты списка перед вставкой
    if ( ListLen == 1 ) { pLast = temp; Reset(); }
    // корректировка текущей позиции - отличие обработки для начала списка
    else if ( CurrPos == 0 ) pCurrLink = temp; else CurrPos++;
    SetRetCode(ListOK);
  }
}             /*-----------------------------------------------*/

void TDatList :: InsLast ( PTDatValue pVal ) { // вставить последним
  PTDatLink temp = GetLink(pVal,pStop);              // SKIP_ON
  if ( temp == NULL ) SetRetCode(ListNoMem);
  else {
    if ( pLast != NULL ) pLast->SetNextLink(temp);
    pLast = temp; ListLen++;
    // проверка пустоты списка перед вставкой
    if ( ListLen == 1 ) { pFirst = temp; Reset(); }
    // корректировка текущей позиции - отличие при pCurrLink за концом списка
    if ( IsListEnded() ) pCurrLink = temp;
    SetRetCode(ListOK);
  }                                                  // SKIP_OFF
}             /*-----------------------------------------------*/

void TDatList :: InsCurrent ( PTDatValue pVal ) { // вставить перед текущим
  if ( IsEmpty() || ( pCurrLink == pFirst ) ) InsFirst(pVal);   // SKIP_ON
  else if ( IsListEnded() ) InsLast(pVal);
  else if ( pPrevLink == pStop ) SetRetCode(ListNoPos);
  else {
    PTDatLink temp = GetLink(pVal,pCurrLink);
    if ( temp == NULL ) SetRetCode(ListNoMem);
    else {
      pCurrLink = temp; pPrevLink->SetNextLink(temp); ListLen++;
      SetRetCode(ListOK);
    }
  }                                                  // SKIP_OFF
}             /*-----------------------------------------------*/

// методы удаления звеньев

void TDatList :: DelFirst ( void ) { // удалить первое звено
  if ( IsEmpty() ) SetRetCode(ListEmpty);            // SKIP_ON
  else {
    PTDatLink temp = pFirst;
    pFirst = pFirst->GetNextDatLink();
    DelLink(temp); ListLen--;
    if ( IsEmpty() ) { pLast = pStop; Reset(); }
    // корректировка текущей позиции - отличие обработки для начала списка
    else if ( CurrPos == 0 ) pCurrLink == pFirst;
    else if ( CurrPos == 1 ) pPrevLink == pStop;
    if ( CurrPos > 0 ) CurrPos--;
    SetRetCode(ListOK);
  }                                                  // SKIP_OFF
}             /*-----------------------------------------------*/
```

```
void TDatList :: DelCurrent ( void ) { // удалить текущее звено
  if ( pCurrLink == pStop ) SetRetCode(ListNoPos);              // SKIP_ON
  else if ( pCurrLink == pFirst ) DelFirst();
  else {
    PTDatLink temp = pCurrLink;
    pCurrLink = pCurrLink->GetNextDatLink();
    pPrevLink->SetNextLink(pCurrLink);
    DelLink(temp); ListLen--;
    // отработка ситуации удаления последнего звена
    if ( pCurrLink == pLast ) { pLast = pPrevLink; pCurrLink = pStop; }
    SetRetCode(ListOK);
  }                                                             // SKIP_OFF
}            /*-------------------------------------------*/

void TDatList :: DelList ( void ) { // удалить весь список
  while ( !IsEmpty() ) DelFirst();
  pFirst  = pLast = pPrevLink = pCurrLink = pStop;
  CurrPos = -1;
}

// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// Copyright (c) Гергель В.П. 09.08.2000
//
// Тестирование списка

#include "tmonom.h"
#include "datlist.h"
#include <iostream.h>
#include <conio.h>

void main() {
  TDatList st;
  TMonom *pVal;
  int temp;
  cout << "Тестирование списка" << endl;
  for ( int i=0; i<5; i++ ) {
    pVal = new TMonom(i,10*i);
    st.InsLast(pVal);
    cout << "Положили в список значение "
         << pVal->GetCoeff() << "; " << pVal->GetIndex()
         << " Код " << st.GetRetCode() << endl;
  }
  // печать списка
  for ( st.Reset(); !st.IsListEnded(); st.GoNext() ) {
    pVal = (TMonom*)st.GetDatValue();
    cout << "Взяли из списка значение "
       << pVal->GetCoeff() << "; " << pVal->GetIndex() << " Код "
       << st.GetRetCode() << endl;
  }
  st.DelList();
  cout << "Нажмите любую клавишу" << endl;
  getch();
}
```

```
// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// tlist.h - Copyright (c) Гергель В.П. 26.07.2002
//
// Списки - шаблон доступа

#ifndef __TLIST_H
#define __TLIST_H

#include "datlist.h"
template <class TDatClass>
class TList : public TDatList {
  public:
    TDatClass GetValue ( TLinkPos mode = CURRENT ) const { // получить значение
      TDatClass *pTemp;
      pTemp =  dynamic_cast<TDatClass *>(GetDatValue(mode)->GetCopy());
      if (pTemp != NULL ) return *pTemp;
      else { SetRetCode(DataErr); TDatClass temp; return temp; }
    }
    void InsValue ( TDatClass &Val, TLinkPos mode = CURRENT ) { // вставить значение
      PTDatValue pVal = Val.GetCopy();
      if ( mode == FIRST ) InsFirst(pVal);
      else if ( mode == LAST ) InsLast(pVal);
      else InsCurrent(pVal);
    }
    void DelValue ( TLinkPos mode = CURRENT ) { // удалить значение
      if ( mode == FIRST ) DelFirst();
      else if ( mode == CURRENT ) DelCurrent();
    }
};
#endif

// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// Copyright (c) Гергель В.П. 09.08.2000
//
// Тестирование списка (работа со списками через класс-переходник)

#include "tmonom.h"
#include "tlist.h"
#include <iostream.h>
#include <conio.h>

void main() {
  TList<TMonom> st;
  TMonom monom;
  cout << "Тестирование списка" << endl;
  for ( int i=0; i<5; i++ ) {
    monom.SetCoeff(i);
    monom.SetIndex(10*i);
    st.InsValue(monom); // очередь
    cout << "Положили в список значение "
         << monom.GetCoeff() << "; " << monom.GetIndex()
         << " Код " << st.GetRetCode() << endl;
  }
  // печать списка
  for ( st.Reset(); !st.IsListEnded(); st.GoNext() ) {
    monom = st.GetValue();
    cout << "Взяли из списка значение "
         << monom.GetCoeff() << "; " << monom.GetIndex()
         << " Код " << st.GetRetCode() << endl;
  }
  st.DelList();
}
```

Нижегородский Государственный Университет им. Н.И. Лобачевского

Общий курс:

Методы программирования - 2

Тема 6:

Разработка системы для арифметических действий над многочленами от нескольких переменных

Гергель В.П., профессор
кафедра МО ЭВМ ВМК

Содержание

2.1. Система для арифметических действий над многочленами

1. Важность рассматриваемого примера

- ☑ Графическое изображения является одной из наиболее предпочтительных форм представления информации для человека. Типовой подход – растровое (дискретное) формирование изображений. Проблемы
  - Большие затраты памяти (полноцветное изображение одного экрана 1000x1000 занимает 3Мб)
  - Сложность масштабирования и трудоемкость обработки
- ↳ Возможный выход -- *векторное (структурное) представление графики*, когда изображения формируются на основе моделей визуализируемых данных (например, прямая может быть задана уравнением линейной функции с двумя числовыми коэффициентами).
- ↳ Упростим пример – рассмотрим визуализацию только функциональных зависимостей на примере полиномов от нескольких переменных
- ☑ Рассматриваемый пример организации обработки полиномов может рассматриваться также как введение в *проблематику аналитических вычислений* на ЭВМ

2.1. Система для арифметических действий над многочленами

2. Постановка задачи

- ☑ Полиномы как формальный объект хорошо изучены в математике. Математическая модель – *алгебра полиномов*.
- ☑ Под *многочленом* понимается выражение из нескольких *термов*, соединенных знаками сложения или вычитания.
- ☑ Терм включает *коэффициент* и *моном*, содержащий одну или несколько переменных, каждая из которых может иметь *степень*

$$P = \Sigma\ Coeff * X^A Y^B Z^C \qquad - \qquad P(X, Y, Z) = 3x^3z - 2y^2z^2 + 3$$

- ☑ В число возможных вычислительных процедур над полиномами входят действия по вычислению значений полинома при заданных значениях переменных, а также большинство известных арифметических операций (сложение, вычитание, вычисление частных производных, интегрирование и т.п.)

2.1. Система для арифметических действий над многочленами

3. Анализ операций для выбора эффективной структуры хранения…

- ☑ Частный случай – полиномы от одной переменной

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x^1 + a_0$$

- ☑ Возможный вариант структуры хранения – стеки
- ☑ В дальнейшем ограничимся рассмотрением *полиномов от трех переменных X, Y, Z.*

2.1. Система для арифметических действий над многочленами

3. Анализ операций для выбора эффективной структуры хранения…

- ☑ Одна из наиболее частных операций – *приведение подобных мономов.*
- ☑ Для ускорения поиска подобных элементов целесообразно ввести какое-либо правило упорядочения мономов (*отношение следования*)
- ☑ Возможный подход -- использование алфавитного порядка для упорядочения слов в словарях (*лексикографический порядок*). Установим старшинство переменных в порядке XYZ. Тогда

$$X^{A1}Y^{B1}Z^{C1} > X^{A2}Y^{B2}Z^{C2} \Leftrightarrow$$

$$(A_1 > A_2) \cup (A_1 = A_2) \& (B_1 > B_2) \cup (A_1 = A_2) \& (B_1 = B_2) \& (C_1 > C_2)$$

- ☑ Установленный порядок является линейный ⇒ правило следования может быть сформулировано в более простом виде

3. Анализ операций для выбора эффективной структуры хранения…

☑ Пусть область возможных значений степеней переменных имеет вид $0 \le A, B, C < 10$

Тогда можно установить взаимно-однозначное соответствие троек (A, B, C) и целых чисел следующим образом

$(A, B, C) \sim ABC = A*100+B*10+C$

Обратное соответствие определяется при помощи выражений A=E(ABC%100), B=E(ABC-A*100)%10, C=ABC-A*100-B*10

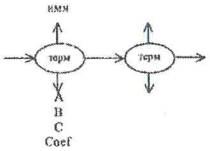
☑ Получаемые по степеням целые величины ABC (будем называть их далее свернутыми степенями или индексами) порождают тот же самый, ранее установленный, порядок следования мономов

$X^{A1}Y^{B1}Z^{C1} > X^{A2}Y^{B2}Z^{C2} \Leftrightarrow ABC_1 > ABC_2$

☑ Приведенные выражения представляют собой правила *позиционной системы счисления* (где N=10 есть основание системы)

3. Анализ операций для выбора эффективной структуры хранения…

☑ Таким образом, на множестве мономов определено отношение следования и многочлен может быть рассмотрен как линейная структура, элементами которой являются термы

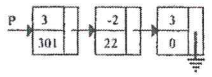


(на схеме полинома термы с нулевыми коэффициентами не указываются)

3. Анализ операций для выбора эффективной структуры хранения

☑ В ходе вычислений количество мономов в полиноме может изменяться ⇒ полиному соответствует динамическая структура

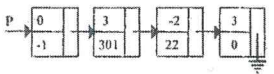
☑ Целесообразной структурой хранения полинома является линейный список



4. Обеспечение однородности структуры хранения полиномов…

☑ Структура хранения полинома, тождественно равного нулю, не содержит звеньев (список вырождается). Данная ситуация может отражаться установкой нулевого значения указателю на список, но тогда все программы для полиномов должны включать специальные действия по обнаружению и обработке этого уникального состояния полинома

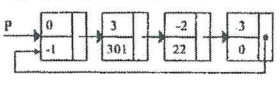
☑ Возможное решение проблемы может состоять во введении дополнительного служебного звена, размещаемого в начале списке (*звено-заголовок*)



(звено-заголовок маркируется логически-недопустимым значением индекса монома)

4. Обеспечение однородности структуры хранения полиномов…

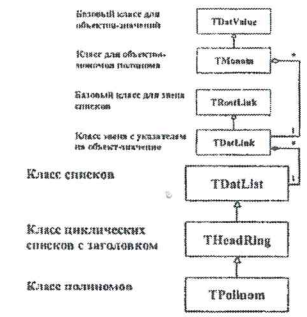
☑ Аналогичным образом можно уйти от проверки нулевого указателя последнего звена, установив в последнем звене в качестве следующего звена первый элемент списка (звено-заголовок). Данная модификация приводит к использованию в качестве структуры хранения *циклический список*



(цикличность списка, кроме того, позволит разработать более эффективные программы обработки полиномов)

☑ Структура хранения нулевого полинома в этом случае имеет вид

5. Проектирование иерархии классов…



2.1. Система для арифметических действий над многочленами

5. Проектирование иерархии классов…

☑ Класс **THeadRing** является производным от класса TDatList. Какие методы надо переопределить ?

- THeadRing – добавить создание звена-заголовка
- ~THeadRing – добавить удаление звена-заголовка
- InsFirst – добавить установку указателя звена-заголовка на pFirst
- DelFirst – добавить установку указателя звена-заголовка на pFirst

2.1. Система для арифметических действий над многочленами

5. Проектирование иерархии классов…

☑ Класс **TPolinom** обеспечивает поддержку структуры хранения и реализацию операций обработки над полиномами

Методы класса:

- TPolinom – конструктор задания полинома по массиву параметров
- GetMonom – получение указателя на моном из текущего звена списка
- operator+ – сложение полиномов
- operator= – присваивание

2.1. Система для арифметических действий над многочленами

5. Проектирование иерархии классов…

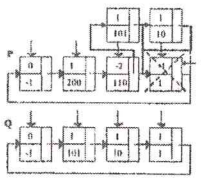
Основные алгоритмические моменты метода сложения полиномов operator+ состоят в следующем:

- результат сложения запоминается в объекте первого операнда;
- операция сложения сводится к последовательной обработке мономов полиномов-операндов p и q:
  - Если моном p меньше монома q, то моном q добавляется в полином p; и текущая позиция в q сдвигается вправо;
  - Если моном p старше монома q, то текущая позиция в p сдвигается вправо;
  - Если моном p равен мономy q, то коэффициенты мономов складываются и запоминаются в p; далее если результат сложения равен 0, то моном в p удаляется и текущая позиция в q сдвигается вправо; если же результат сложения ненулевой, то текущая позиция сдвигается вправо и в p и в q.

2.1. Система для арифметических действий над многочленами

5. Проектирование иерархии классов…

Пусть $P = x^2 - 2xy - z$ и $Q = x + y + z$. Рассмотрим последовательность действий, выполняемых при сложении этих полиномов



Пример: программа, приложение

Заключение

- Использование структур данных для представления на ЭВМ математических моделей
- Основные принципы разработки структуры хранения полиномов (лексикографическое упорядочение мономов, свертка степеней при помощи правил позиционной системы счисления, однородность структуры хранения, циклический список)
- Поэтапная разработка программ через проектирование иерархии классов

Вопросы для обсуждения

- Допустимость расширения диапазона возможных значений степеней
- Возможность применения разработанных программ для представления произвольного индексированного набора значений (разреженных матриц !?)

Темы заданий для самостоятельной работы

- Разработка дополнительных вычислительных операций для полиномов (вычисление значения полинома при заданных значениях переменных, вычисление частных производных, интегрирование и др.)

Следующая тема

- Структуры хранения и обработка текстовой информации

```
// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// Copyright (c) Гергель В.П. 27.08.2000
//
// Циклические списки с заголовком

#ifndef __HEADRING_H
#define __HEADRING_H

#include "datlist.h"

class THeadRing : public TDatList{
  protected:
    PTDatLink pHead;      // заголовок, pFirst - звено за pHead
  public:
    THeadRing ();
   ~THeadRing ();
   // вставка звеньев
    virtual void InsFirst( PTDatValue pVal=NULL ); // вставить после заголовка
    // удаление звеньев
    virtual void DelFirst( void );                 // удалить первое звено
};
#endif
```

```
// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// Copyright (c) Гергель В.П. 27.08.2000
//
// Циклические списки с заголовком

#include "HeadRing.h"

THeadRing :: THeadRing () : TDatList() {
  InsLast(); pHead = pFirst; ListLen = 0;
  pStop = pHead; Reset();
  pFirst->SetNextLink(pFirst);
}              /*-------------------------------------------*/

THeadRing ::  ~THeadRing () {
  TDatList::DelList();
  DelLink(pHead);
  pHead = NULL;
}              /*-------------------------------------------*/

void THeadRing :: InsFirst ( PTDatValue pVal ) { // вставить после заголовка
  TDatList::InsFirst(pVal);
  if ( RetCode == DataOK ) pHead->SetNextLink(pFirst);
}              /*-------------------------------------------*/

void THeadRing :: DelFirst  ( void ) {           // удалить первое звено
  TDatList::DelFirst(); pHead->SetNextLink(pFirst);
}              /*-------------------------------------------*/
```

```
// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// polinom.h - Copyright (c) Гергель В.П. 09.08.2000
//
// Полиномы

#ifndef __POLINOM_H
#define __POLINOM_H

#include "tmonom.h"
#include "headring.h"
class TPolinom : public THeadRing {
  public:
    TPolinom ( int monoms[][2]=NULL, int km=0 );
    TPolinom ( const TPolinom &q);       // конструктор копирования
    PTMonom  GetMonom()  { return (PTMonom)GetDatValue(); }
    TPolinom & operator+( TPolinom &q); // сложение полиномов
    TPolinom & operator=( TPolinom &q); // присваивание
    friend   ostream& operator<<(ostream &os,TPolinom &q);
};
#endif
```

```
// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// polinom.cpp - Copyright (c) Гергель В.П. 09.08.2000
//
// Полиномы

#include "polinom.h"

TPolinom :: TPolinom ( int monoms[][2], int km ) {
  PTMonom pMonom = new TMonom(0,-1);
  pHead->SetDatValue(pMonom);
  for ( int i=0; i<km; i++ ) {
    pMonom = new TMonom(monoms[i][0],monoms[i][1]);
    InsLast(pMonom);
  }
}              /*-------------------------------------------*/

TPolinom & TPolinom :: operator + ( TPolinom &q) { // сложение полиномов
  PTMonom  pm, qm, rm;
  Reset();  q.Reset();
  while (1) {
    pm = GetMonom(); qm = q.GetMonom();
    if ( pm->Index < qm->Index ) {
      // моном pm младше монома qm => добавление монома qm в полином p
      rm = new TMonom(qm->Coeff,qm->Index);
      InsCurrent(rm);  q.GoNext();
    }
    else if ( pm->Index > qm->Index ) GoNext();
    else { // индексы мономов равны (но это могут быть головы !)
      if ( pm->Index == -1 ) break;
      pm->Coeff += qm->Coeff;
      if ( pm->Coeff != 0 ) { GoNext(); q.GoNext(); }
      else { // удаление монома с нулевым коэффициентом
        DelCurrent(); q.GoNext();
      }
    }
  }
  return *this;
}              /*-------------------------------------------*/
```

```
TPolinom :: TPolinom ( const TPolinom &q) {
  PTMonom pMonom = new TMonom(0,-1);
  pHead->SetDatValue(pMonom);
  for ( q.Reset(); !q.IsListEnded(); q.GoNext() ) {
    PTMonom pMonom = q.GetMonom();
    InsLast(pMonom->GetCopy());
  }
}              /*-------------------------------------------*/

TPolinom & TPolinom :: operator = ( TPolinom &q) { // присваивание
  DelList();
  for ( q.Reset(); !q.IsListEnded(); q.GoNext() ) {
    PTMonom pMonom = q.GetMonom();
    InsLast(pMonom->GetCopy()); //delete pMonom;
  }
  return *this;
}              /*-------------------------------------------*/

ostream& operator<<(ostream &os,TPolinom &q) {
  for ( q.Reset(); !q.IsListEnded(); q.GoNext() )
    cout << *q.GetMonom() << endl;
  return os;
}             /*-------------------------------------------*/

// ННГУ, ВМК, Курс "Методы программирования-2", C++, ООП
//
// Copyright (c) Гергель В.П. 09.08.2000
//
// Тестирование программ работы с полиномами

#include <conio.h>
#include "polinom.h"
#include <iostream.h>

void main() {
  cout << "Тестирование полиномов" << endl;
  int ms1[][2] = { { 1, 543 }, { 3, 432 }, { 5, 321 }, { 7, 210 }, { 9, 100 } };
  int mn1 = sizeof(ms1) / ( 2 * sizeof(int) );
  TPolinom p(ms1,mn1);
  cout << "1 полином" << endl << p;
  int ms2[][2] = { { 2, 643 }, { 4, 431 }, {-5, 321 }, { 8, 110 }, {10,  50 } };
  int mn2 = sizeof(ms2) / ( 2 * sizeof(int) );
  TPolinom q(ms2,mn2);
  cout << "2 полином" << endl << q;
  TPolinom r=p+q;
  cout << "Полином-результат" << endl << r;
  cout << "Нажмите любую клавишу" << endl;
  getch();

}
```

Вопросы для контроля по общему курсу "ЭВМ и программирование"

Введение

1. Проблема доказательства правильности программ
2. Способы снижения сложности программного обеспечения

Тема 1. Структуры действия и структуры данных

1. Рекурсивное описание вычислительного процесса и структуры данных.
2. Структуры данных и математические структуры.
3. Переменные структуры и схемы структуры.
4. Понятие экземпляра, схемы структуры.
5. Линейные структуры данных
6. Структура машинной памяти. Вектор памяти как образ линейной структуры.
7. Практическая работа 1: Структура хранения множеств
8. Практическая работа 2: Структуры хранения для матриц специального вида
9. Динамические структуры.
10. Практическая работа 3: Структуры хранения динамических структур типа стек
11. Практическая работа 4: Структуры хранения динамических структур типа очередь
12. Сравнение структур хранения линейных и динамических структур.
13. Статическое и динамическое распределение памяти.
14. Управление памятью путем перепаковки структур хранения.
15. Практическая работа 5: Структура хранения нескольких стеков в общей памяти.
16. Роль гипотез о росте структур при разработке систем управления памятью путем перепаковки.
17. Оценка параметров модели в ходе выполнения программ (адаптация).
18. Линейный список.
19. Способы реализации списков на языках высокого уровня.
20. Управление свободной памятью при использовании сцепления.
21. Практическая работа 6: Реализация структуры хранения нескольких стеков с использованием списков на языке высокого уровня.
22. Сравнение непрерывной и списковой структур хранения.
23. Динамическое распределение памяти в языке C/C++ (выделение и освобождение памяти).
24. Реализация стека с использованием динамически распределяемой памяти.
25. Пример использования стеков: поразрядная сортировка.
26. Пример использования стеков: преобразование арифметических выражений в польскую форму записи.
27. Практическая работа 7: Разработка общего представления линейного списка для обеспечения списковой структуры хранения.
28. Общая характеристика стандартной библиотеки шаблонов.

Тема 2. Динамические структуры и конструирование математических моделей.

1. Система для арифметических действий над полиномами (представление полиномов, управление памятью, выполнение операций).
2. Представление многочленов от нескольких переменных. Исключение хранения мономов с нулевыми коэффициентами.
3. Схема наследования программ для обеспечения структуры хранения полиномов.
4. Реализация программ для обеспечения работы с линейным циклическим списком.
5. Структура класса для представления на ЭВМ полиномов от нескольких переменных.
6. Алгоритм сложения многочленов от нескольких переменных.
7. Представление текста связным списком.
8. Операторы объединения списков и расчленения списка.
9. Алгоритм обхода иерархического списка.
10. Копирование списка.
11. Сборка мусора.
12. Плексы как представление рисунков, состоящих из точек и соединяющих их отрезков.
13. Алгоритм обхода плекса.
14. Алгоритм вставки линии.
15. Плекс, как представление арифметического выражения.

Подписано в печать 02.09.2016 г. Формат 60×84 1/8.
Бумага офсетная. Печать цифровая.
Усл. печ. л. 1,2. Заказ № 831. Тираж 200 экз.

Отпечатано с предоставленных материалов
в типографии ННГУ им. Н.И. Лобачевского.
603000, г. Нижний Новгород, ул. Б. Покровская, 37