ПЕРМСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
Кафедра Информационных технологий и
автоматизированных систем

Викентьева О. Л., Гусин А. Н., Полякова О. А.

Методические указания к курсовой работе по дисциплине "Алгоритмические языки и программирование" для студентов специальности АСУ

Пермь 2005

Оглавление

Введение

Курсовая работа предназначена для отработки навыков программирования задач средней сложности у студентов дневного и заочного отделения специальностей "Автоматизированные системы управления" в рамках дисциплины "Алгоритмические языки и программирование".

Целью курсовой работы является закрепление и углубление знаний, полученных студентами в курсе "Алгоритмические языки и программирование", развитие навыков при выборе представления исходных данных, использовании объектно-ориентированного подхода при написании программ на языке C++, тестировании и отладки программы, оформлении документации на программную разработку.

1. Порядок выполнения курсовой работы

Курсовая работа по курсу "Алгоритмические языки и программирование" выполняется индивидуально каждым студентом в соответствии с выданным преподавателем вариантом. Обязательным является использование в курсовой работе объектно-ориентированного подхода и пользовательских классов. Курсовая работа выполняется в среде MS Visual C++6.0.

В процессе работы автор должен

1. Выполнить анализ предметной области.
2. Разработать пользовательские классы.
3. Разработать алгоритмы, реализующие компонентные функции классов и перегруженные операции.
4. Разработать пользовательский интерфейс для ввода и получения информации.
5. Предусмотреть обработку исключительных ситуаций, возникающих во время работы программы.
6. Провести отладку и тестирование программы.
7. Оформить для нее документацию.

Все этапы работы должны быть отражены в пояснительной записке.

Таблица 1 Примерный график выполнения работы

№	Этап курсовой работы	Неделя семестра
1	Анализ предметной области, декомпозиция системы на компоненты (классы, объекты, модули).	1-3
2	Разработка пользовательских классов.	4-5
3	Разработка алгоритмов, используемых при решении задачи.	6-8
4	Разработка интерфейса.	9
5	Отладка и тестирование программы.	10-12
6	Оформление пояснительной записки.	13
7	Защита курсовой работы.	14-15

2. Варианты заданий

2.1. Калькуляторы

1) Разработать класс «Калькулятор», выполняющий указанные в варианте операции для заданных исходных данных. Программа должна выполнять ввод данных, проверку правильности введенных данных, выдачу соответствующих сообщений в случае возникновения ошибок.

2)Протокол работы калькулятора записать в файл. Протокол должен включать исходные данные, введенные пользователем, выполняемые операции и результаты их выполнения. В случае возникновения ошибки в файл записывается соответствующее сообщение.

3)Предусмотреть возможность просмотра этого файла из программы калькулятора.

В варианте указаны вид данных, обрабатываемых калькулятором, и операции, выполняемые калькулятором.

Таблица 2 Калькуляторы

Вариант	Вид данных	Операции
1	Двадцатипятизначные числа.	Сложение, вычитание, умножение, деление, целочисленное деление, остаток от деления, отмена последней операции, сброс результата.
2	Обыкновенные и десятичные дроби.	Сложение, вычитание, умножение, деление, преобразование десятичной дроби в обыкновенную и обратно, отмена последней операции, сброс результата.
3	Комплексные числа в алгебраической, тригонометрической и экспоненциальной формах.	Сложение, вычитание, умножение, деление, возведение в целую степень, извлечение квадратного корня (по формуле Муавра), преобразование из одной формы в другую, отмена последней операции, сброс результата.
4	Даты в российском, американском и английском форматах.	- дата1-дата2=кол-во месяцев - дата1-дата2=кол-во недель - дата1-дата2=кол-во дней

		- дата1-дата2=кол-во часов - дата1-дата2=кол-во минут - дата1-дата2=кол-во секунд - дата1-месяцы=дата2 - дата1-недели=дата2 - дата1-дни=дата2 - дата1-часы=дата2 - дата1-минуты=дата2 - дата1-секунды=дата2 - дата1+месяцы=дата2 - дата1+недели=дата2 - дата1+дни=дата2 - дата1+часы=дата2 - дата1+минуты=дата2 - дата1+секунды=дата2 Обратить внимание на проверку правильности вводимой информации (33 июня не бывает)
5	Обыкновенные дроби в двух формах: - числитель/ знаменатель; - целая часть числитель/знаменатель.	Сложение, вычитание, умножение, деление, возведение в целую степень, сокращение дроби, обмен местами числителя и знаменателя дроби, смена знака, отмена последней операции, сброс результата.
6	Квадратные уравнения (в действительных или комплексных числах)	Вычисление корней в форме, заданной пользователем.
7	Углы, измеряемые в градусной системе мер (1 градус=60 минут, 1 минута=60 секунд)	Сложение, вычитание, умножение, деление, нахождение тригонометрических и обратных тригонометрических

		функций, изменение знака угла.
8	Выражения со скобками	Вычисление заданного выражения со скобками, проверка правильности заданного выражения.
9	Целые и вещественные числа в различных системах счисления, основания систем счисления взять следующие: 2, 3, 8, 9, 10, 16.	Перевод чисел из одной системы счисления в другую.
10	Логарифмы чисел при различных основаниях.	Сложение, вычитание, умножение, деление, возведение в степень, переход к другому основанию.
11	Многочлены (до 8-ой степени, задаются коэффициенты).	Сложение, вычитание, умножение, деление, возведение в целую степень, подстановка многочлена вместо переменной в другой многочлен, вычисление значения многочлена.
12	Система уравнений с тремя неизвестными (задаются коэффициенты).	Ввод, хранение уравнений в памяти (до 4 систем), вычисление неизвестных.
13	Алгебраические или трансцендентные уравнения (задаются функции, интервалы, на которых есть корень, точность вычислений).	Решение уравнения методами половинного деления, Ньютона или итераций (в зависимости от выбора пользователя).
14	Интервалы времени (задаются в часах, минутах, секундах).	Ввод данных, ввод только в часах, минутах или секундах, нахождение величины временного интервала, суммы, разности, преобразование интервала в часы, минуты, секунды.
15	Римские цифры.	Сложение, вычитание, умножение, целочисленное

		деление, остаток от целочисленного деления, перевод из римской системы счисления в десятичную и обратно, отмена последней операции, сброс результата.

2.2. Динамические структуры данных

Разработать класс, реализующий указанную в задании структуру данных. Разработать интерфейс программы, позволяющий вводить данные и получать результаты в удобной для пользователя форме. Программа должна проверять правильность вводимых данных. В случае возникновения ошибок должны выдаваться сообщения. Должна быть предусмотрена возможность получения данных из файла и запись результатов файл.

В варианте указаны вид данных и операции, выполняемые с указанной структурой.

Таблица 3 Структуры данных

Варант	Вид данных	Операции
16	Бинарное дерево	Создать идеально сбалансированное дерево, распечатать его (в виде дерева), найти величину наибольшего элемента дерева, напечатать элементы из всех листьев дерева, найти длину пути от корня до ближайшей вершины с элементом E.
17	Бинарное дерево	Создать идеально сбалансированное дерево, распечатать его (в виде дерева), определить максимальную глубину о дерева, определить есть ли в дереве хотя бы 2 одинаковых элемента, определить входит ли элемент E в дерево, если таких элементов несколько, то определить число вхождений элемента E в дерево.
18	Бинарное дерево	Создать идеально сбалансированное дерево T, распечатать его (в виде дерева), построить и напечатать по уровням дерево поиска T1 для дерева T, удалить из непустого дерева T все четные элементы, найти количество вершин на n-ом уровне непустого дерева T (корень - вершина 0 уровня).

19	Линейный список	Составить программу, которая содержит информацию о наличие автобусов в автобусном парке. Сведения об автобусе содержат: - номер автобуса; - фамилию и инициалы водителя; - номер маршрута. Программа должна обеспечивать: - формирование данных обо всех автобусах в виде списка; - формирование списка автобусов выехавших из парка; - формирование списка автобусов оставшихся в парке; - вывод сведений об автобусах находящихся на маршруте и об автобусах оставшихся в парке.
20	Бинарное дерево	Составить программу, которая содержит текущую информацию о книгах в библиотеке. Сведения о книгах содержат: - номер УДК; - фамилию и инициалы автора; - название; - год издания; - количество экземпляров в библиотеке. Программа должна обеспечить : - начальное формирование данных обо всех книгах в виде бинарного дерева; - добавление данных о книгах, поступающих в библиотеку; - удаление данных о списываемых книгах; - выдачу данных о наличии в библиотеке требуемой книги.
21	Линейный список	Составить программу, которая содержит текущую информацию о книгах в библиотеке. Сведения о книгах содержат: - номер УДК; - Фамилию и инициалы автора; - Название; - Год издания; - Количество экземпляров в библиотеке. Программа должна обеспечить: - начальное формирование данных обо всех книгах в виде линейного списка;

		- добавление данных о книгах, поступающих в библиотеку; - изменение данных при вводе информации о том, что пользователь берет или возвращает книгу; - выдаче данных о наличии книг в библиотеке.
22	Линейный список	Составить программу, которая моделирует заполнение гибкого диска. В процессе работы файлы могут записываться на диск и удаляться с него. Файлы могут иметь произвольную длину. Если при удалении образовался свободный участок, то вновь записываемый файл помещается на этом свободном участке, либо, если он не помещается на этом участке, то его следует разместить после последнего записанного файла. Если файл превосходит длину самого большого участка, выдается аварийное сообщение. Рекомендуется создать список свободных участков и список занятых участков памяти на диске.
23	Бинарное дерево	Составить программу, которая формирует англо-русский словарь. Словарь должен содержать английское слово, русское слово и количество обращений к слову. Программа должна: - обеспечить начальный ввод словаря(по алфавиту) с конкретными значениями счетчиков обращений; - формирует новое дерево, в котором слова отсортированы не по алфавиту, а по количеству обращений. Пользователь должен иметь возможность добавлять новые слова, удалять существующие, выполнять писк нужного слова, выполнять просмотр обоих вариантов словаря
24	Список	Составить программу, которая содержит информацию о квартирах, содержащихся в базе данных бюро обмена квартир. Сведения о каждой квартире содержат: - количество комнат; - общую площадь;

		- этаж; - адрес; Программа должна обеспечить: - формирование картотеки; - ввод заявки на обмен; - поиск подходящего варианта (при равенстве количества комнат и этажа и различии площадей в пределах 10% выводится соответствующая карточка, а сами сведения удаляются из списка, в противном случае поступившая заявка включается в список); - Вывод всего списка.
25	Бинарное дерево	Автоматизированная информационная система на железнодорожном вокзале содержит сведения об отправлении поездов дальнего следования. Для каждого поезда указывается: - номер поезда; - станция назначения; - время отправления. Составить программу, которая 1. обеспечивает первоначальный ввод данных в информационную систему и формирование дерева; 2. производит вывод всего дерева; 3. вводит номер поезда и выводит данные об этом поезде; 4. вводит название станции назначения и выводит данные о всех поездах, которые следуют до этой станции.

3. Декомпозиция системы на компоненты

Объектная методология предлагает рассматривать предметную область и проектировать программную систему как совокупность взаимодействующих друг с другом объектов. Объект обладает состоянием, поведением и идентичностью; структура и поведение схожих объектов определяет общий для них класс. Состояние объекта характеризуется перечнем всех свойств данного объекта и текущими значениями каждого из этих свойств. К числу свойств относятся присущие объекту или приобретаемые им характеристики, черты, качества или способности, делающие данный объект самим собой. Перечень свойств объекта является, как правило, статическим, поскольку эти свойства составляют неизменяемую основу объекта. Их принято называть атрибутами класса.

Поведение – это то, как объект действует и реагирует. Поведение объекта определяется выполняемыми над ним операциями и его состоянием, причем некоторые операции имеют побочное действие: они изменяют состояние. Объектно-ориентированный стиль программирования связан с воздействием на объекты путем передачи

им сообщений (т.е. вызывая методы, описанные в их классе). Операция над объектом порождает некоторую реакцию этого объекта. Операции, которые можно выполнить по отношению к данному объекту, и реакция объекта на внешние воздействия определяют поведение этого объекта.

Операция – это услуга, которую может предоставить класс. На практике над объектами выполняются операции пяти видов.

Таблица 4 Операции над объектами

Конструктор	имеет то же имя, что и класс; определяет способ создания объекта или его инициализации
Деструктор	операция, выполняющая очистку памяти, когда объект класса выходит за пределы области видимости или он удаляется; имеет то же имя, что и класс со знаком «~» перед ним.
Модификатор	операция, которая изменяет состояние объекта
Селектор	операция, считывающая состояние объекта, но не меняющая состояния
Итератор	операция, позволяющая организовать доступ ко всем частям объекта в строго определенной последовательности

Объекты могут создаваться различными способами. Некоторые объекты являются локальными переменными, другие глобальными, третьи – членами классов и т.д.

Между объектами могут существовать различные отношения:

- ассоциация;
- наследование;
- агрегация;
- зависимость;
- и др.

Отношения двух любых объектов основываются на предположениях, которыми один обладает относительно другого: об операциях, которые можно выполнять и об ожидаемом поведении. Связь – это специфическое сопоставление, через которое один объект (клиент) запрашивает услугу у другого объекта (сервера) или через которое один объект находит путь к другому. Она дает классу возможность узнавать об атрибутах, операциях и связях другого класса. В нотации языка UML взаимодействие между классами отражают связывающими их линиями. Чтобы один класс мог послать сообщение другому, между ними должна существовать связь.

Ассоциация – это смысловая связь, которая не имеет направления и не объясняет, как классы общаются друг с другом. Однако именно это требуется на ранней стадии анализа, поэтому мы фиксируем только участников, их роли и мощность отношения. В дальнейшем она, как правило, конкретизируется и принимает вид одного из рассматриваемых далее отношений. На диаграммах UML эту связь отображают обыкновенной линией, связывающую классы:

рис. 1. Связь между класами

Наследование – это такое отношение между классами, когда один класс повторяет структуру и поведение другого класса (одиночное наследование) или других (множественное наследование) классов. Класс, структура и поведение которого наследуются, называется суперклассом. Производный от суперкласса класс называется подклассом. Это означает, что наследование устанавливает между классами иерархию общего и частного. Подкласс обычно расширяет или ограничивает существующую структуру и поведение своего суперкласса.

Связи наследования также называют обобщениями (generalization) и изображают в виде больших белых стрелок от класса-потомка к классу-предку.

Помимо наследуемых, каждый подкласс имеет свои собственные уникальные атрибуты, операции и связи.

Лучше всего строить иерархию классов, в которой от общих классов с помощью наследования образуются более специализированные.

рис. 2. Отношение наследования

Агрегация. В то время как связи обозначают равноправные или «клиент-серверные» отношения между объектами, агрегация описывает отношения целого и части, приводящие к соответствующей иерархии объектов, причем, идя от целого (агрегата) мы можем придти к его частям (атрибутам). В этом смысле агрегация – специализированный частный случай ассоциации. Объект, являющийся атрибутом другого объекта (агрегата), имеет связь со своим агрегатом. Через эту связь агрегат может посылать ему сообщения.

Агрегации отображают в виде линии с ромбиком у класса, являющегося целым:

рис. 3 Отношение агрегации

В дополнение к простой агрегации UML вводит более сильную разновидность агрегации, называемую композицией. Согласно композиции объект-часть может принадлежать только единственному целому, и, кроме того, зачастую жизненный цикл частей совпадает с циклом целого: любое удаление целого распространяется на его части. Такой вид агрегации отображается с помощью закрашенного ромбика со стороны целого:

рис. 4. Отношение композиции

Зависимость – отношение, при котором один класс пользуется услугами другого класса. Отношения зависимости между классами изображают в виде стрелки, проведенной пунктирной линией:

рис. 5. Отношение зависимости

Объявления классов и функций принято отделять от реализации последних. Хорошим стилем считается, когда объявления собраны в заголовочный файл с расширением .h, а реализации – в файле с расширением .cpp (имена файлов одинаковые).

Рассмотрим декомпозицию системы на компоненты на примере калькулятора, выполняющего операции с датами в виде дд.мм.гг. Этот калькулятор выполняет сложение или вычитание двух дат и выдает результат в виде новой даты или количества дней (месяцев, недель), содержащихся в новой дате.

В этой системе можно выделить классы «Дата»,«Калькулятор» и «Диалог с пользователем».Объекты классов обладают свойствами, индивидуальностью и поведением. Свойства выражаются в виде атрибутов, индивидуальность – в виде значений соответствующих атрибутов, поведение - в виде методов, описанных в соответствующем классе.

Для выполнения арифметического действия Калькулятор должен знать две даты и знак операции. Следовательно, класс Калькулятор должен включать три объекта класса Дата (регистры): для операнда_1, операнда_2 и результата. Над операндами выполняется какая-то арифметическая операция, знак этой операции тоже можно хранить в отдельной переменной класса Калькулятор. Результат

калькулятор выдает в виде одной из четырех форм: дата, количество дней, недель или месяцев. Таким образом, мы определили, что класс Калькулятор будет содержать следующие компонентные данные:

```
date op1,op2,rez;//операнды и результат
int znak;//код операции
int form;//код формы для вывода результата
```

Калькулятор необходимо инициализировать, следовательно, класс Калькулятор должен содержать конструктор. Кроме того, Калькулятор должен содержать модификаторы для изменения операндов, знака и формы вывода, селектор для получения результата и метод, который будет выполнять вычисление результата в зависимости от знака операции. Таким образом, мы получаем следующее описание класса Calc:

```
class Calc
{
Date op1;//операнд 1
Date op2;//операнд 2
Date res;//результат
int znak;//знак операции
int form;//форма вывода результата
public:
Calc();//конструктор
void set_op1(date);//модификатор для операнда 1
void set_op2(date); //модификатор для операнда 2
Date get_res();//селектор для получения результата
void set_znak(int); //модификатор для знака операции
void set_form(int); //модификатор для формы вывода
void execute();//вычисление результата
};
```

В классе Calc используются переменные типа Date – данные, с которыми работает Калькулятор. Между классами Calc и Date существует отношение агрегации, причем, это будет не просто агрегация, а композиция, т. к. операнды не могут существовать отдельно от калькулятора.

Рассмотрим свойства класса Дата. Над Датой можно выполнять операции, Дату надо вводить и выводить. Ввод и вывод Даты проще всего выполнять в виде строки. Но для выполнения операций с датами надо предусматривать представление Даты в виде чисел. Операции по сложению и вычитанию дат можно выполнять по следующему алгоритму:

1. Перевести обе даты в дни.
2. Выполнить операцию.
3. Перевести результат в требуемую форму (дата, дни, недели, месяцы и т. п.)

Таким образом, класс Date должен иметь следующие компонентные данные:

```
int d;//количество дней в дате
char* s;//представление даты в виде строки
```

Дата вводится в форме строки, а затем вычисляется, сколько дней она содержит. Для преобразования Даты из одной формы в другую надо предусмотреть компонентные функции:

```
void days_to_string(); - переводит дни в строку
```

void string_to_days(); - переводит строку в дни

Над числами выполняются арифметические операции, следовательно, удобно будет перегрузить эти операции в виде компонентных операций-функций:

Date operator -(Date&D);

Date operator +(Date D);

Date& operator =(const Date &D);

Ввод и вывод чисел удобно реализовать с помощью глобальных перегруженных операций-функций >> и <<.

Таким образом, описание класса предназначенного для работы с датами будет выглядеть следующим образом:

```
class date
{
char *date_string;//строка для хранения даты в виде дд.мм.гггг.
int days;//количество дней в дате для выполнения расчетов
public:
date();//конструктор
date(const date&);//конструктор копирования
~date();//деструктор
//инициализация даты: заполняет атрибут date_string, вычисляет
//количество дней в этой дате, затем заполняет атрибут days
void init(char*);
// вспомогательная функция, которая по количеству дней формирует
//строку вида дд.мм.гггг.
void days_to_string();
// вспомогательная функция для вычисления дней в дате
void string_to_days();
//перегруженная функция для сложения двух дат
date operator+(date&);
//перегруженная функция для сложения двух дат
date operator-(date&);
//перегруженная функция присваивания
date& operator=(const date&);
//дружественная функция для ввода даты
friend istream&operator>>(istream&in,date&);
//дружественная функция для вывода даты
friend ostream&operator<<(ostream&in,date&);
};
```

Класс «Диалог с пользователем» осуществляет ввод данных в виде строк, заносит эти строки в операнды калькулятора, вводит операцию, заносит ее в знак операции калькулятора, вводит форму, требуемого результата, заносит ее в соответствующий атрибут калькулятора, получает результат и выводит его в нужной форме. Эти действия повторяются до тех пор, пока пользователь не введет команду для окончания расчетов. Для ввода данных можно использовать меню. Таким образом, класс Диалог может иметь следующую реализацию:

```
class Dialog
{
public:
Calc c;//калькулятор для выполнения расчетов
void menu();//меню для диалога с пользователем
```

```
//цикл, в котором осуществляется вывод меню, ввод данных,
//вычисление результата и проверка окончания расчетов
void doing();
};
```

Основная функция будет иметь вид:

```
void main()
{
     Dialog d;
     d.doing();
}
```

Этот пример был рассмотрен для того,чтобы упростить работу по декомпозиции классов при выполнении курсовой работы. Но ни в коем случае не следует считать, что выполнить работу можно только таким способом!

4. Методические указания

1. Курсовая работа выполняется в среде Visual C++ 6.0 как консольное приложение.
2. При выполнении курсовой работы обязательным является использование объектно-ориентированного программирования.
3. Как правило, класс как тип, определенный пользователем, должен содержать скрытые поля и следующие функции
   - Конструкторы, определяющие, как инициализируются объекты класса;
   - Набор методов, реализующих свойства класса (методы, возвращающие значения скрытых полей класса описываются с модификатором const, для того, чтобы не изменялись значения полей);
   - Набор операций, позволяющий копировать, присваивать, сравнивать объекты и производить с ними требуемые действия;
   - Класс исключений, используемый для сообщений об ошибках с помощью генерации исключительных ситуаций.
4. В курсовой работе должно использоваться не менее 3 классов (см. п.4), причем диалог с пользователем должен быть реализован как отдельный класс.
5. Каждый класс должен быть реализован в виде двух файлов: заголовочного (.h), содержащего описание класса и файла (.cpp), содержащего реализацию методов класса. Основная функция main реализуется в виде отдельного файла. Если в работе используются глобальные функции, они также должны быть размещены в отдельном файле.
6. В курсовой работе должны использоваться перегруженные функции-операции для выполнения заданных в варианте операций. Например, для добавления элемента в список можно перегрузить операцию сложения (+) или инкремент (++).
7. Для реализации протокола (варианты 1-15) и а также записи данных в файл и получения данных из файла (варианты 16-25) использовать файловые потоки.
8. Предусмотреть проверку корректности данных. При проверке использовать механизм исключительных ситуаций.

5. Рекомендации по программированию

- При создании класса следует хорошо продумать его интерфейс – средства работы с классом для тех программ, которые будут его

использовать. Интерфейс должен быть интуитивно понятным и включать только методы. Поля данных класса должны быть скрытыми.

- Не следует определять методы типа get/set для всех скрытых полей класса — это все равно, что открыть к ним доступ, только более сложным способом. Поля класса вводятся только для того, чтобы реализовать свойства класса, представленные в его интерфейсе с помощью методов.
- Не нужно расширять интерфейс класса без необходимости, «на всякий случай», поскольку увеличение количества методов ведет к трудности понимания класса пользователем. В идеале интерфейс должен быть полным, то есть предоставлять возможность выполнить любые разумные действия с классом, и минимальным — без дублирования и пересечения возможностей методов.
- В виде методов рекомендуется определять только действия, реализующие свойства класса. Если какое-либо действие можно реализовать, не обращаясь к скрытым полям класса, его нет необходимости описывать как метод; лучше описать его как обычную функцию, поместив ее в общее с классом пространство имен. Если функция выполняет действие, не являющееся свойством класса, но нуждается в доступе к его скрытым полям, ее следует объявить как дружественную. Но в общем случае дружественных функций и классов надо избегать, поскольку главной идеей ООП является минимизация связей между инкапсулированными классами.
- Для увеличения производительности программы наиболее часто вызываемые методы можно объявить как встроенные (inline). В основном это касается коротких методов, тело которых оказывается меньше размера кода, генерируемого для их вызова. Кроме ускорения программы за счет исключения вызовов, это дает возможность компилятору производить более полную оптимизацию. Однако необходимо учитывать, что директива inline носит для компилятора рекомендательный характер.
- Конструкторы и деструкторы делать встраиваемыми не рекомендуется, поскольку в них фактически присутствует код, помещаемый компилятором, размер этого кода может быть весьма значительным (например, в конструкторе производного класса должны быть вызваны конструкторы всех базовых и вложенных классов).
- Перегруженные операции класса должны иметь интуитивно понятный общепринятый смысл (например, не следует заставлять операцию + выполнять что-либо, кроме сложения или добавления).
- Если какая-либо операция перегружена, следует, если возможно, перегрузить и аналогичные операции, например, +, += и ++ (компилятор этого автоматически не сделает). При этом операции должны иметь ту же семантику, что и их стандартные аналоги.
- И конструктор копирования, и операция присваивания, создаваемые по умолчанию, выполняют поэлементное копирование из области-источника в область-приемник. Если объект содержит указатели, это приведет к тому, что после копирования два соответствующих указателя разных объектов будут ссылаться на одну и ту же область памяти. При уничтожении первого из объектов эта память будет освобождена, а повторная попытка освободить ее при уничтожении второго объекта приведет к

неопределенному поведению программы. Поэтому для классов, содержащих поля-указатели, следует всегда явно определять конструктор копирования и операцию присваивания, выполняющие выделение памяти под динамические поля объекта.

- Динамическая память, выделенная в конструкторе объекта, должна освобождаться в его деструкторе. Невыполнение этого требования приводит к утечкам памяти. Удаление нулевого указателя безопасно (при этом ничего не происходит), поэтому если конструкторы, конструкторы копирования и операция присваивания написаны правильно, любой указатель либо ссылается на выделенную область памяти, либо равен нулю, и к нему можно применять delete без проверки.
- Разница между конструктором копирования и операцией присваивания заключается в том, что последняя работает в том случае, когда объект-приемник уже существует, поэтому в ней перед выделением динамической памяти следует освободить память занятую ранее. Из этого следует, что при реализации операции присваивания для классов, содержащих поля-указатели, необходимо проводить проверку на самоприсваивание и в этом случае оставить объект без изменений. Необходимо также помнить о том, что операция присваивания должна возвращать ссылку на константу.
- В конструкторах для задания начальных значений полям рекомендуется использовать инициализацию, а не присваивание. Инициализация более универсальна, так как может применяться в тех случаях, когда присваиванием пользоваться нельзя (например, при задании значений константным полям или ссылкам). Кроме того, она выполняется более эффективно, потому что создание объекта в C++ начинается с инициализации его полей конструктором по умолчанию, после чего выполняется вызываемый конструктор. Необходимо учитывать и тот факт, что поля инициализируются в порядке их объявления, а не в порядке появления в списке инициализации. Поэтому для уменьшения числа возможных ошибок порядок указания полей в списке инициализации конструктора должен соответствовать порядку их объявления в классе.
- Статические поля не должны инициализироваться в конструкторе, поскольку им нужно присваивать начальное значение только один раз для каждого класса, а конструктор выполняется для каждого объекта класса. Статические поля инициализируются в глобальной области определения (вне любой функции).
- Конструкторы копирования также должны использовать списки инициализации полей, поскольку иначе для базовых классов и вложенных объектов будут вызваны конструкторы по умолчанию.
- Операция присваивания не наследуется, поэтому она должна быть определена в производных классах. При этом из нее следует явным образом вызывать соответствующую операцию базового класса
- Открытое наследование класса Y из класса X означает, что Y представляет собой разновидность класса X, то есть более конкретную, частную концепцию. Базовый класс X является более общим понятием, чем Y. Везде, где можно использовать X, можно использовать и Y, но не наоборот (вспомните, что на место

ссылок на базовый класс можно передавать ссылку на любой из производных). Необходимо помнить, что во время выполнения программы не существует иерархии классов и передачи сообщений объектам базового класса из производных — есть только конкретные объекты классов, поля которых формируются на основе иерархии на этапе компиляции.

- Методы, которые должны иметь все производные классы, но которые не могут быть реализованы на уровне базового класса, должны быть виртуальными. Например, все объекты иерархии должны уметь выводить информацию о себе. Поскольку она хранится в различных полях производных классов, эту функцию нельзя реализовать в базовом классе. Естественно назвать ее во всех классах одинаково и объявить как виртуальную с тем, чтобы другие методы базового класса могли вызывать ее в зависимости от фактического типа объекта, с которым они работают. По этой причине деструкторы объявляются как виртуальные.
- При переопределении виртуальных методов нельзя изменять наследуемое значение аргумента по умолчанию, поскольку по правилам C++ оно определяется типом указателя, а не фактическим типом объекта, вызвавшего метод.

6. Среда программирования Visual C++ 6.0

6.1. Общий вид окна

Проект (project) – это набор файлов, которые совместно используются для создания одной программы.

Рабочее пространство (workspace) может включать в себя несколько проектов.

После запуска VC++ 6.0 на экране появится окно (рис. 1). Окно содержит:

- Главное меню (1) – список основных команд VC++;
- Панель инструментов (2) - панель с кнопками команд Visual C++;
- Панель рабочего пространства Workspace (3) - содержит две вкладки:
  - ClassView – отображает список классов в проекте,
  - FileView – отображает список файлов, входящих в проект.
- Окно для редактирования кодов (4) – окно с текстом программы;
- Выходную панель результатов компиляции (5) - окно для вывода сообщений в процессе компиляции или отладки, показывает текущую стадию компиляции, список ошибок и предупреждений и их количество.

Рис. 1. Окно VC++ 6.0.

6.2. Создание консольного приложения и работа с ним

Консольное приложение – это приложение, которое с точки зрения программиста является программой DOS, но может использовать всю доступную оперативную память (если каждый элемент данных программы не будет превышать 1 Мб). Этот тип приложения запускается в особом окне, которое называется “Окно MS-DOS”. На примере консольных приложений прослеживаются этапы развития VC++ при переходе от одной версии к другой.

Каждое приложение, разрабатываемое как отдельный проект в среде VC++6.0, нуждается в том, чтобы ему соответствовало свое собственное рабочее пространство. Рабочее пространство включает в себя те папки, в которых будут храниться файлы, содержащие информацию о конфигурации проекта. Для того чтобы создать новое пространство для проекта, надо выполнить следующие действия:

1. В линейке меню нажать на меню **File.**
2. Выбрать пункт **New** или нажать **Ctrl+N.**
3. Появится окно **New.** В нем содержится четыре вкладки: Files, Projects, Workspaces, Other Documents. Выбрать вкладку Projects.
4. Из списка возможных проектов выбрать **Win32 Console Application** для создания приложения DOS.
5. В поле **Project name** ввести имя проекта.
6. В поле **Location** ввести путь для размещения каталога проекта, или, нажав на кнопку справа **[…]**, выбрать нужную директорию.
7. Должен быть установлен флажок **Create New Workspace.** Тогда будет создано новое рабочее окно. Нажать кнопку **OK**
8. Установить один из флажков:

- An empty project – создается пустой проект, не содержащий заготовок для файлов;

- A simple application – создается простейшая заготовка, состоящая из заголовочного файла StdAfx.h, файла StdAfx.cpp и файла реализации;

- A "Hello World" application и **An application that supports MFC** являются демонстрационными и разными способами демонстрируют вывод на экран строки символов.

Нажать кнопку **Finish**. Появится информация о созданном проекте содержащая: тип проекта, некоторые особенности и директорию.

После создания проекта в него необходимо записать код программы. При этом можно создать новый файл или добавить в проект существующий файл.

Для создания нового файла надо выполнить следующие действия:

1. Выбрать меню **File > New** или **Project > Add to Project > New**.
2. Открыть вкладку **Files**.
3. Выбрать **C++ Source File**.
4. Чтобы создаваемый файл был автоматически присоединен к проекту, необходимо установить флаг **Add to project**.
5. В поле **Filename** ввести имя файла.
6. В поле **Location** указать путь для создания файла.
7. Нажать **OK**.

Для добавления существующего файла надо:

1. Выбрать в меню **File > Add to Project > Files**
2. Указать полное имя файла, который нужно присоединить

Для открытия существующего проекта надо:

1. Выбрать меню **File > Open Workspace**
2. Указать файл с расширением **.dsw**

Для сохранения текущего проекта надо выбрать в главном меню **File > Save Workspace.**

Для закрытия текущего проекта надо выбрать в главном меню **File > Close Workspace.**

После создания или открытия проекта в окне **Workspace** появится или список классов, или список файлов входящих в проект. В зависимости от типа проекта, он будет или пустой, или содержать изначально некоторые файлы, присущие данному типу. Проект приложения для DOS изначально пустой. В него можно добавить новые файлы или присоединить уже существующие.

6.3. Компиляция и запуск проекта

Для компиляции проекта надо выбрать в главном меню **Build > Build <имя проекта>** или нажать клавишу F7.

Visual C++ 6.0 откомпилирует исходные файлы и создаст соответствующие файлы с расширением .obj. Затем эти файлы соединяются в исполняемый файл. Весь процесс компиляции и создания исполняемого файла отображается в окне Output, вкладка Build. После компиляции файла его можно запустить.

Для запуска исполняемого файла надо выбрать в главном меню **Build > Execute <имя файла>.exe** или нажмите клавиши **Ctrl+F5** . Если файл был создан, то он запустится. Для повторного запуска файла не нужно его снова компилировать. Но если в программу были внесены изменения, то перед запуском необходимо выполнить компиляцию. Выполняется именно файл с расширением .exe, а не

текущий проект, т.е. в процессе запуска компиляции не происходит.

6.4. Отладка программы

Для отладки программы используется команда главного меню **Build>Start Debug> Step Into** – отладка с заходом в функции, которая начинается с первой строки функции main или **Build>Start Debug> Run to Cursor** – выполнение программы до курсора, т. е. отладка начинается с той строки, в которой установлен курсор. После выполнения этой команды выполнение программы происходит в режиме отладчика. Переход к следующей строке программы можно выполнять с помощью команды **Step Into (F11)** (с заходом во все вызываемые функции) или с помощью команды **Step over (F10**) (без захода в вызываемые функции). Выход из функции нижнего уровня выполняется командой **Step Out (Shift+F11).** Текущие значения переменных можно просматривать:

1) в специальных окнах **Watch** (отображает значения всех используемых переменных) и **Value (**отображает значения заданных пользователем переменных);
2) при наведении курсора мышки на переменную отображается текущее значение этой переменной.

6.5. Создание рабочего пространства для нескольких проектов

Несколько проектов можно объединить в одно рабочее пространство с помощью команды **Project/Insert Project into Work-space**. Активный проект, т. е. тот, который будет выполняться, устанавливается с помощью команды **Project/Set Active Project.** Активный проект надо отметить галочкой.

7. Содержание отчета

1. Титульный лист с указанием фамилии, группы, исполнителя и варианта курсовой работы.
2. Постановка задачи.
3. Диаграмма классов.
4. Описание пользовательских классов (название класса, компонентные данные и их назначение, компонентные функции и их назначение).
5. Реализация компонентных функций.
6. Реализация основной программы.
7. Примеры экранных форм или диалогов, используемые для взаимодействия с пользователем.
8. Примеры результатов работы программы.
9. Программная документация:
   - названия файлов, из которых состоит проект;
   - названия библиотек, необходимых для выполнения проекта;
   - инструкция пользователю для работы с проектом.
10. Список используемой литературы;
11. Листинг программы.

8. Список литературы

1. Вирт Н. Алгоритмы + структуры данных = программы . – М. : Мир, 1985. – 406 с.
2. Голуб А. И. С и С++. Правила программирования. – М.: БИНОМ, 1996. – 272 с.

3. Ноткин А. М. Динамические структуры данных. Учебное пособие. - Пермь, 1994, -66с.

4. Павловская Т. А. С/С++. Программирование на языке высокого уровня. – СПб.: Питер, 2001. – 464с.

5. Павловская Т. А., Щупак Ю. А. С++. Объектно-ориентированное программирование: Практикум. – СПб.:Питер, 2004. – 265 с.

6. Подбельский В. В. Язык Си++ : Учеб. пособие.– М.: Финансы и статистика, 1996.–560с.

7. Подбельский В. В., Фомин С. С. Программирование на языке Си: Учеб. пособие. – М.:Финансы и статистика, 1998.–600с.

8. Страуструп Б. Язык программирования С++. –СПб.:БИНОМ, 1999. – 991 с.

9. Страуструп Б. Язык программирования Си++: Пер. с англ. – М.: Радио и связь, 1991.-352с.