Лабораторная работа № 1

КЛАССЫ И ОБЪЕКТЫ В С++

Цель. Получить практические навыки реализации классов на С++.

Основное содержание работы.

Написать программу, в которой создаются и разрушаются объекты, определенного пользователем класса. Выполнить исследование вызовов конструкторов и деструкторов.

Краткие теоретические сведения.

Класс.

Класс – фундаментальное понятие С++, он лежит в основе многих свойств С++. Класс предоставляет механизм для создания объектов. В классе отражены важнейшие концепции объектно-ориентированного программирования: инкапсуляция, наследование, полиморфизм.

С точки зрения синтаксиса, класс в С++ – это структурированный тип, образованный на основе уже существующих типов.

В этом смысле класс является расширением понятия структуры. В простейшем случае класс можно определить с помощью конструкции:

тип_класса имя_класса{список_членов_класса};

где

тип_класса – одно из служебных слов class, struct, union;

имя_класса – идентификатор;

список_членов_класса – определения и описания типизированных данных и принадлежащих классу функций.

Функции – это методы класса, определяющие операции над объектом.

Данные – это поля объекта, образующие его структуру. Значения полей определяет состояние объекта.

Примеры.

```
struct date                    // дата
{int month,day,year;            // поля: месяц, день, год
 void set(int,int,int);       // метод – установить дату
 void get(int*,int*,int*);     // метод – получить дату
 void next();                 // метод – установить следующую дату
 void print();                 // метод – вывести дату
};
struct class complex          // комплексное число
{double re,im;
 double real(){return(re);}
 double imag(){return(im);}
 void set(double x,double y){re = x; im = y;}
 void print(){cout<<"re = "<<re; cout<<"im = "<<im;}
};
```

Для описания объекта класса (экземпляра класса) используется конструкция

```
имя_класса имя_объекта;
date today,my_birthday;
date *point = &today;          // указатель на объект типа date
date clim[30];                // массив объектов
date &name = my_birthday;     // ссылка на объект
```

В определяемые объекты входят данные, соответствующие членам – данным класса. Функции – члены класса позволяют обрабатывать данные конкретных объектов класса. Обращаться к данным объекта и вызывать функции для объекта можно двумя способами. Первый с помощью "квалифицированных" имен:

```
имя_объекта. имя_данного
имя_объекта. имя_функции
```

Например:

```
complex x1,x2;
x1.re = 1.24;
x1.im = 2.3;
x2.set(5.1,1.7);
x1.print();
```

Второй способ доступа использует указатель на объект
указатель_на_объект->имя_компонента

```
complex *point = &x1; // или point = new complex;
point ->re = 1.24;
point ->im = 2.3;
point ->print();
```

Доступность компонентов класса.

В рассмотренных ранее примерах классов компоненты классов являются общедоступными. В любом месте программы, где “видно” определение класса, можно получить доступ к компонентам объекта класса. Тем самым не выполняется основной принцип абстракции данных – инкапсуляция (сокрытие) данных внутри объекта. Для изменения видимости компонент в определении класса можно использовать спецификаторы доступа: public, private, protected.

Общедоступные (public) компоненты доступны в любой части программы. Они могут использоваться любой функцией как внутри данного класса, так и вне его. Доступ извне осуществляется через имя объекта:

имя_объекта.имя_члена_класса
ссылка_на_объект.имя_члена_класса
указатель_на_объект->имя_члена_класса

Собственные (private) компоненты локализованы в классе и не доступны извне. Они могут использоваться функциями – членами данного класса и функциями – “друзьями” того класса, в котором они описаны.

Защищенные (protected) компоненты доступны внутри класса и в производных классах.

Изменить статус доступа к компонентам класса можно и с помощью использования в определении класса ключевого слова class. В этом случае все компоненты класса по умолчанию являются собственными.

Пример.

```
class complex
{
  double re, im;            // private по умолчанию
  public:
  double real(){return re;}
  double imag(){return im;}
  void set(double x,double y){re = x; im = y;}
};
```

Конструктор.

Недостатком рассмотренных ранее классов является отсутствие автоматической инициализации создаваемых объектов. Для каждого вновь создаваемого объекта необходимо было вызвать функцию типа set (как для класса complex) либо явным образом присваивать значения данным объекта. Однако для инициализации объектов класса в его определение можно явно включить специальную компонентную функцию, называемую конструктором. Формат определения конструктора следующий:

имя_класса(список_форм_параметров){операторы_тела_конструктора}
Имя этой компонентной функции по правилам языка C++ должно совпадать с именем класса. Такая функция автоматически вызывается при определении или размещении в памяти с помощью оператора new каждого объекта класса.

Пример.

```
complex(double re1 = 0.0,double im1 = 0.0){re = re1; im = im1;}
```

Конструктор выделяет память для объекта и инициализирует данные – члены класса.

Конструктор имеет ряд особенностей:

• Для конструктора не определяется тип возвращаемого значения. Даже тип void не допустим.

•Указатель на конструктор не может быть определен, и соответственно нельзя получить адрес конструктора.

•Конструкторы не наследуются.

•Конструкторы не могут быть описаны с ключевыми словами virtual, static, const, mutuable, valatile.

Конструктор всегда существует для любого класса, причем, если он не определен явно, он создается автоматически. По умолчанию создается конструктор без параметров и конструктор копирования. Если конструктор описан явно, то конструктор по умолчанию не создается. По умолчанию конструкторы создаются общедоступными (public).

Параметром конструктора не может быть его собственный класс, но может быть ссылка на него (T&). Без явного указания программиста конструктор всегда автоматически вызывается при определении (создании) объекта. В этом случае вызывается конструктор без параметров. Для явного вызова конструктора используются две формы:

имя_класса имя_объекта (фактические_параметры);

имя_класса (фактические_параметры);

Первая форма допускается только при не пустом списке фактических параметров. Она предусматривает вызов конструктора при определении нового объекта данного класса:

complex ss (5.9,0.15);

Вторая форма вызова приводит к созданию объекта без имени:

complex ss = complex (5.9,0.15);

Существуют два способа инициализации данных объекта с помощью конструктора. Ранее мы рассматривали первый способ, а именно, передача значений параметров в тело конструктора. Второй способ предусматривает применение списка инициализаторов данного класса. Этот список помещается между списком параметров и телом конструктора. Каждый инициализатор списка относится к конкретному компоненту и имеет вид:

имя_данного (выражение)

Примеры.

```
class CLASS_A
{
  int i; float e; char c;
public:
  CLASS_A(int ii,float ee,char cc) : i(8),e( i * ee + ii ),c(cc){}
...
};
```

Класс “символьная строка”.

```
#include <string.h>
#include <iostream.h>
class string
{
  char *ch; // указатель на текстовую строку
  int len;     // длина текстовой строки
public:
  // конструкторы
  // создает объект – пустая строка
  string(int N = 80): len(0){ch = new char[N+1]; ch[0] = '\0';}
  // создает объект по заданной строке
  string(const char *arch){len = strlen(arch);
      ch = new char[len+1];
      strcpy(ch,arch);}
  // компоненты-функции
  // возвращает ссылку на длину строки
  int& len_str(void){return len;}
```

```
        // возвращает указатель на строку
        char *str(void){return ch;}
      . . .};
```

Здесь у класса string два конструктора – перегружаемые функции.

По умолчанию создается также конструктор копирования вида T::T(const T&), где T – имя класса. Конструктор копирования вызывается всякий раз, когда выполняется копирование объектов, принадлежащих классу. В частности он вызывается:

а) когда объект передается функции по значению;

б) при построении временного объекта как возвращаемого значения функции;

в) при использовании объекта для инициализации другого объекта.

Если класс не содержит явным образом определенного конструктора копирования, то при возникновении одной из этих трех ситуаций производится побитовое копирование объекта. Побитовое копирование не во всех случаях является адекватным. Именно для таких случаев и необходимо определить собственный конструктор копирования. Например, в классе string:

```
        string(const string& st)
        {len=strlen(st.len);
        ch=new char[len+1];
        strcpy(ch,st.ch); }
```

Можно создавать массив объектов, однако при этом соответствующий класс должен иметь конструктор по умолчанию (без параметров).

Массив объектов может инициализироваться либо автоматически конструктором по умолчанию, либо явным присваиванием значений каждому элементу массива.

```
class demo{
int x;
public:
demo(){x=0;}
demo(int i){x=i;}
};
void main(){
class demo a[20]; //вызов конструктора без параметров(по умолчанию)
class demo b[2]={demo(10),demo(100)};//явное присваивание
```

Деструктор.

Динамическое выделение памяти для объекта создает необходимость освобождения этой памяти при уничтожении объекта. Например, если объект формируется как локальный внутри блока, то целесообразно, чтобы при выходе из блока, когда уже объект перестает существовать, выделенная для него память была возвращена. Желательно, чтобы освобождение памяти происходило автоматически. Такую возможность обеспечивает специальный компонент класса – деструктор класса. Его формат:

```
~имя_класса(){операторы_тела_деструктора}
```

Имя деструктора совпадает с именем его класса, но предваряется символом “~” (тильда).

Деструктор не имеет параметров и возвращаемого значения. Вызов деструктора выполняется не явно (автоматически), как только объект класса уничтожается.

Например, при выходе за область определения или при вызове оператора delete для указателя на объект.

```
        string *p=new string “строка”);
        delete p;
```

Если в классе деструктор не определен явно, то компилятор генерирует деструктор по умолчанию, который просто освобождает память, занятую данными объекта. В тех случаях, когда требуется выполнить освобождение и других объектов памяти, например область, на которую указывает ch в объекте string, необходимо определить деструктор явно:

```
~string(){delete []ch;}
```

Так же, как и для конструктора, не может быть определен указатель на деструктор.

Указатели на компоненты-функции.
Можно определить указатель на компоненты-функции.
тип_возвр_значения(имя_класса::*имя_указателя_на_функцию)(специф_параметров_функции);
Пример.

```
double(complex : :*ptcom)(); // Определение указателя
ptcom = &complex : : real; // Настройка указателя
// Теперь для объекта A можно вызвать его функцию
complex A(5.2,2.7);
cout<<(A.*ptcom)();
```

Можно определить также тип указателя на функцию

```
typedef double&(complex::*PF)();
```

а затем определить и сам указатель

```
PF ptcom=&complex::real;
```

Порядок выполнения работы.

1. Определить пользовательский класс в соответствии с вариантом задания (смотри приложение).
2. Определить в классе следующие конструкторы: без параметров, с параметрами, копирования.

3. Определить в классе деструктор.
4. Определить в классе компоненты-функции для просмотра и установки полей данных.
5. Определить указатель на компоненту-функцию.
6. Определить указатель на экземпляр класса.
7. Написать демонстрационную программу, в которой создаются и разрушаются объекты пользовательского класса и каждый вызов конструктора и деструктора сопровождается выдачей соответствующего сообщения (какой объект какой конструктор или деструктор вызвал).
8. Показать в программе использование указателя на объект и указателя на компоненту-функцию.

Методические указания.

1. Пример определения класса.

```
const int LNAME=25;
class STUDENT{
char name[LNAME];             // имя
int age;                      // возраст
float grade;                   // рейтинг
public:
STUDENT();                    // конструктор без параметров
STUDENT(char*,int,float);      // конструктор с параметрами
STUDENT(const STUDENT&); // конструктор копирования
~STUDENT();
char * GetName() ;
int GetAge() const;
float GetGrade() const;
void SetName(char*);
void SetAge(int);
void SetGrade(float);
void Set(char*,int,float);
void Show(); };
```

Более профессионально определение поля name типа указатель: char* name. Однако в этом случае реализация компонентов-функций усложняется.

2. Пример реализации конструктора с выдачей сообщения.

```
STUDENT::STUDENT(char*NAME,int AGE,float GRADE)
{
strcpy(name,NAME); age=AGE; grade=GRADE;
```

```
      cout<< \nКонструктор с параметрами вызван для объекта
<<this<<endl;
      }
```
3. Следует предусмотреть в программе все возможные способы вызова конструктора копирования. Напоминаем, что конструктор копирования вызывается:

а) при использовании объекта для инициализации другого объекта

Пример.

```
STUDENT a(“Иванов”,19,50), b=a;
```

б) когда объект передается функции по значению

Пример.

```
void View(STUDENT a){a.Show;}
```

в) при построении временного объекта как возвращаемого значения функции

Пример.

```
STUDENT NoName(STUDENT & student)
{STUDENT temp(student);
temp.SetName(“NoName”);
return temp;}

STUDENT c=NoName(a);
```

4. В программе необходимо предусмотреть размещение объектов как в статической, так и в динамической памяти, а также создание массивов объектов.

Примеры.

а) массив студентов размещается в статической памяти

```
STUDENT gruppa[3];
gruppa[0].Set(“Иванов”,19,50);
```

и т.д.

или

```
STUDENT gruppa[3]={STUDENT(“Иванов”,19,50),
                   STUDENT(“Петрова”,18,25.5),
                   STUDENT(“Сидоров”,18,45.5)};
```

б) массив студентов размещается в динамической памяти

```
STUDENT *p;
p=new STUDENT [3];
p-> Set(“Иванов”,19,50);
```

и т.д.

5. Пример использования указателя на компонентную функцию

```
void (STUDENT::*pf)();
pf=&STUDENT::Show;
(p[1].*pf)();
```

6. Программа использует три файла:

- заголовочный h-файл с определением класса,
- cpp-файл с реализацией класса,
- cpp-файл демонстрационной программы.

Для предотвращения многократного включения файла-заголовка следует использовать директивы препроцессора

```
#ifndef STUDENTH
#define STUDENTH
// модуль STUDENT.H
...
#endif
```

Содержание отчета.

1. Титульный лист: название дисциплины; номер и наименование работы; фамилия, имя, отчество студента; дата выполнения.

2. Постановка задачи. Следует дать конкретную постановку, т.е. указать, какой класс должен быть реализован, какие должны быть в нем конструкторы, компоненты-функции и т.д.

3. Определение пользовательского класса с комментариями.

4. Реализация конструкторов и деструктора.

5. Фрагмент программы, показывающий использование указателя на объект и указателя на функцию с объяснением.

6. Листинг основной программы, в котором должно быть указано, в каком месте и какой конструктор или деструктор вызываются.

Приложение. Варианты заданий.

Описания членов - данных пользовательских классов

1. СТУДЕНТ
имя – char*
курс – int
пол – int(bool)

2. СЛУЖАЩИЙ
имя – char*
возраст – int
рабочий стаж – int

3. КАДРЫ
имя – char*
номер цеха – int
разряд – int

4. ИЗДЕЛИЕ
имя – char*
шифр – char*
количество – int

5. БИБЛИОТЕКА
имя – char*
автор – char*
стоимость – float

6. ЭКЗАМЕН
имя студента – char*
дата – int
оценка – int

7. АДРЕС
имя – char*
улица – char*
номер дома – int

8. ТОВАР
имя – char*
количество – int
стоимость – float

9. КВИТАНЦИЯ
номер – int
дата – int
сумма – float

10. ЦЕХ
имя – char*
начальник – char*
количество
работающих – int

11. ПЕРСОНА
имя – char*
возраст – int
пол – int(bool)

12. АВТОМОБИЛЬ
марка – char*
мощность – int
стоимось – float

13. СТРАНА
имя – char*
форма
правления – char*
площать – float

14. ЖИВОТНОЕ
имя – char*
класс – char*
средний вес – int

15. КОРАБЛЬ
имя – char*
водоизмещение – int
тип – char*