ЛАБОРАТОРНАЯ РАБОТА № 12

Простейшие классы и объекты

1. **Цель работы** : Получить практические навыки реализации классов на C++. Изучить основные способы работы с пользовательским типом данных «класс», его объектами, методами и способы доступа к ним.

2. Краткие теоретические сведения

2.1 ***Класс*** – это определённый пользователем ***тип***. Объявление класса задаёт *представление объектов этого класса* и *набор операций*, которые можно применять к таким объектам. Класс обеспечивает абстракцию данных, он скрывает детали представления объекта и предоставляет доступ к содержащимся в нём данным только посредством функций и операций, описанных как часть этого класса.

2.2 ***Объявление класса***

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. Функция, объявленная в классе *функцией-членом класса*.. Функции – члены класса позволяют обрабатывать данные объектов класса.

Данные – это **поля объекта**, образующие его структуру(свойства). Значения полей определяет состояние объекта

Пример:

```
class X                                  // Объявление класса X
{ private:
int n;
public:
void set_n(int n) { this->n = n; }
int get_n() const { return n; }
void f();
};
void X::f()                              // Определение функции f из класса X
{ n++; }
Int main() {…
X a, b;                                  // Объявление переменных класса X
a.f();                  // Вызов функции f применяется к переменной a. Таким образом,
…..}                    // изменяется член n переменной a. Переменная b остаётся без изменений.
```

2.3 Доступность компонентов класса.

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, но они могут использоваться функциями – членами данного класса.
Защищенные (protected) компоненты доступны внутри класса и в производных классах.

2.4 Конструктор.

Недостатком рассмотренного ранее класса является отсутствие автоматической инициализации создаваемых объектов. Для каждого вновь создаваемого объекта необходимо определить и вызывать функцию типа set (метод, инициализирующий элементы данных), 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 объекта в памяти с помощью оператора new. Конструктор выделяет память для объекта и инициализирует данные – члены класса.

Конструктор всегда существует для любого класса, причем, если он не определен явно, он создается автоматически общедоступным (public). По умолчанию создается конструктор без параметров.
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.

2.5 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. Такую возможность обеспечивает специальный компонент класса – **деструктор** класса:

имя_класса(){операторы_тела_деструктора}

Если в классе деструктор не определен явно, то компилятор генерирует его по умолчанию только для освобождения памяти, занятой данными объекта. Описывать в классе деструктор явным образом требуется в случае, когда объект содержит указатели на память, выделяемую динамически – иначе при уничтожении объекта память, на которую ссылались его поля-указатели, не будет помечена как свободная.

3. ЗАДАНИЕ на лабораторную работу с методическими указаниями .

1. Определить **пользовательский класс** в соответствии с заданием.
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 поле **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<< \n"Constructor with parameters was called"
<<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
```

4. Варианты заданий.

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

2. ТОВАР
наименование – char*
марка – char*
количество – int

3. АДРЕС
город – char*
улица – char*
номер дома – int

4. ЦЕХ
название – char*
ФИО начальника – char*
количество работающих – int

5. СТРАНА
название – char*
форма правления – char*
площадь – float

6.СЛУЖАЩИЙ
ФИО – char*
возраст – int
рабочий стаж – float

7. БИБЛИОТЕКА
Название издания – char*
автор – char*
стоимость – float

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

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

10. ЖИВОТНОЕ
кличка – char*
класс – char*
средний вес – int

11. КАДРЫ
ФИО – char*
номер цеха – int
разряд – int

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

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

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

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

16. СПОРТСМЕН
вид спорта – char*
дата рождения – char*
разряд – int

17. НЕДВИЖИМОСТЬ
район – char*
тип – char*
площадь – float

18. БАГАЖ
наименование – char*
количество – int
вес – float

19. СОТРУДНИК
ФИО – char*
образование – char*
табельный номер – float

20. СКЛАД
наименование изделия – char*
масса – float
количество – int