Практическое занятие №1.
Тема: Проектирование классов и создание объектов. Перегрузка функций

Цель: получение практических навыков разработки классов и формирования объектов с помощью перегруженных конструкторов.

Общие сведения о классах

Ключевым понятием ООП является объект. Под **объектом** понимается переменная особого типа. Этот особый тип в Си++ создан на базе типа данных **структура** (struct) и называется **класс** (class). *Класс* отличается от *структуры* в языке С++ тем, что в него помимо собственно данных различных типов входят также функции, обрабатывающие эти данные. Класс – это тип, группирующий данные и функции в единое целое с точки зрения их последующей обработки. Объявленная переменная типа class есть объект. Такое объединение данных и обрабатывающих их функций называется *инкапсуляцией.* Она защищает данные от внешнего вмешательства или неправильного использования. В то же время объект может оснащаться средствами программной связи с другими объектами.

Данные и функции, входящие в класс, называются *представителями* или *членами* этого класса. Данные класса ещё называют полями, а функции – методами класса. Члены класса разделяются на закрытые (private) и открытые (public). К закрытым членам обращение возможно только внутри класса и из других частей программы они недоступны. Обычно, таким образом защищаются данные от внешнего вмешательства. К открытым членам класса может производиться обращение извне.

Синтаксис объявления *класса* похож на синтаксис объявления *структуры*. Пример объявления.

```
#include <iostream >
using namespace std;
class myclass                    // класс под именем myclass
{
  int a;                         // закрытая переменная
public:                          // ключевое слово (для открытых членов)
     void set_a(int n);          // функции – открытые члены класса -
      int get_a();               // для доступа к закрытой переменной
};
```

Массивы объектов

Объекты – это переменные, и они имеют те же возможности и признаки, что и переменные любых других типов. Поэтому объекты могут объединяться в массивы. Синтаксис объявления массива объектов совершенно аналогичен

тому, который используется для объявления массива переменных любого другого типа. То же касается и доступа к элементам массива.

Пример с массивом объектов.

```
#include<iostream>
using namespace std;
class one                    //  имя класса
{
  int a;                     // закрытая переменная
public:
  void set_a(int n){a=n;}// функция доступа к закрытой переменной
  int get_a(){return a;} // функция доступа к закрытой переменной
};
  int main( )                 // главная функция
  {
     one obj[8];              // создание массива из 8-и объектов
     int i;
     for(i=0; i<8; i++)       // инициализация переменных объектов
          obj[i].set_a(i);
     for(i=0;i<8;i++)         // вывод значений переменных объектов
       cout<<obj[i].get_a()<<' ';
  return 0;
  }
```

В данном примере объявлен класс **one**, в котором имеется закрытая переменная и две функции доступа к ней. В главной функции создан массив из 8-и объектов, в котором циклической операцией производится инициализация закрытой переменной каждого элемента массива с помощью функции **set_a( ).** Во второй циклической операции производится вывод на экран значения переменной каждого объекта путем использования функции **get_a( ).**

Конструкторы и деструкторы

Среди членов класса могут быть две особенные функции. Их особенность заключается в том, что обращение к ним производится по умолчанию. Первая функция называется *конструктор*. Она вызывается автоматически при создании объекта. Её назначение состоит в том, чтобы присвоить начальные значения (инициализировать) переменным объекта. Вторая функция называется *деструктор*. Она служит для ликвидации последствий использования объекта (например, для освобождения памяти). Поэтому она автоматически вызывается при удалении объекта. Конструктор имеет то же имя, что и класс, частью которого он является. Имя деструктора также совпадает с именем класса, но перед ним ставится символ ~ (тильда). Конструктор и деструктор не возвращают никаких значений.

```
#include<iostream>
using namespace std;
class abc            // объявление класса abc
{
   int a;          // закрытая переменная
public:
     abc();         // конструктор
    ~abc();         // деструктор
   void show();
};
abc::abc()           // описание конструктора
{ cout<<"I am constructor \n";  a=10;}
abc::~abc()          // описание деструктора
{ cout<<"I am destructor \n";}
void abc::show()     // описание функции-члена класса
{ cout<<a<<"\n";}
int main()
{
     abc obj;        // создание объекта
     obj.show();
return 0;
}
```

Перегрузка функций

Если две или более функций имеют одинаковое имя, то говорят, что они перегружены. В этом случае функции должны отличаться типом и (или) числом своих аргументов. Перегрузить функцию не сложно – просто следует объявить и определить все требуемые варианты. На компилятор возлагается задача выбора соответствующей конкретной версии вызываемой функции (а значит и метода обработки данных). Рассмотрим пример перегрузки функции.

```
#include <iostream>
using namespace std;

void date (char * date) {cout <<date<< "\n";}
void date (int month, int day, int year)
{cout <<day<< "/" <<month<<"/" <<year<<"\n";}
int main ( )
{
   date ("11/12/2020");
   date (11, 12, 2020);
return 0;
}
```

В данном примере функция ***date()*** перегружается для получения даты либо в виде строки, либо в виде трех целых чисел. Данный пример показывает,

что перегрузка функции может производиться и вне объектно-ориентированного подхода.

Перегрузка конструкторов

Конструктор – это функция, поэтому он допускает перегрузку. Более того перегрузка конструкторов является общепринятой практикой, т.к. позволяет сделать более гибкой инициализацию закрытых переменных объектов.

В примере с классом **abc** (см. выше) конструктор не содержал параметров. Но обычно параметры присутствуют, т.к. основное назначение конструкторов – производить инициализацию переменных объекта.

В следующем примере в классе **black** применяются два конструктора (конструктор по умолчанию и конструктор с параметрами) и показано, как использовать варианты перегруженного конструктора.

```
#include<iostream>
class black          // объявление класса black
{
     int a,b;     // закрытые переменные
public:
     black()   // конструктор без параметров
     {  a=0;   b=0;}
     black(int x, int y)   // конструктор с параметрами
     {  a=x;   b=y;}
     void show()             // функция- член класса
     {
        cout<<a<<'\t'<<b;
     }
};

int main()
{
  black ob1;         // использование конструктора по умолчанию
  black ob2(20,100);  // использование конструктора с параметрами
  ob1.show();
  ob2.show();
return 0;
}
```

В примере создаются два объекта. Для первого вызывается конструктор без параметров, для второго – с параметрами. Функция **show()** демонстрирует значения закрытых переменных каждого объекта.

Передача объектов функциям

Объекты можно передавать функциям в качестве аргументов точно так же, как передаются данные других типов. Для этого параметр функции

объявляется типом class, а в качестве аргумента при вызове функции используется объект этого класса. Рассмотрим пример.

```
#include<iostream>
using namespace std;
class simple                    // имя класса
{ int i;
  public:
   simple(int n)  {i=n;}  // конструктор
   int get()    { return i;}
};
int sqr(simple obj) // функция – не член класса, которой
{ return obj.get()*obj.get();}  // передается объект
int main()
{
     simple ob1(2),ob2(10);   // создание двух объектов
     cout<<sqr(ob1)<<"  "<<sqr(ob2);
return 0;
}
```

В этом примере объявляется класс simple, который содержит одну переменную, конструктор и функцию, возвращающую значение переменной. Функция **sqr( )** в качестве параметра имеет объект типа **simple**, получает из него значение переменной **i** и возвращает её квадрат. Здесь необходимо отметить, что функция **sqr( )** не является членом класса. В главной функции создаются два объекта с инициализацией. На экран выводится значение квадрата переменной каждого объекта: 4 100.

Важная деталь при передаче объектов состоит в том, что когда объект передается функции в качестве аргумента, обычный конструктор не вызывается. Вместо него вызывается конструктор копии, который по умолчанию создает побитовую (идентичную) копию этого объекта. Но когда эта копия разрушается (обычно при выходе за пределы области видимости по завершении функции), обязательно вызывается деструктор.

Возвращение объектов функциями

Возвращение объектов функциями производится так же, как и значений других типов данных. Для этого необходимо объявить функцию так, чтобы её возвращаемое значение имело имя типа класса, а в операторе **return** указать имя возвращаемого объекта. Возвращенный объект должен быть скопирован в месте вызова функции в объект такого же класса; другими словами, операция присваивания может производиться только между объектами одного класса. Рассмотрим пример.

```
#include <iostream>
using namespace std;
```

```
class simple
{
int i;
public:
   simple(int n)  // конструктор
   {  i=n; }
   ~simple() { }
   int get()    { return i;}
};

simple input()  // функция возвращает объект типа simple
{
     simple locob(25);  // создание объекта типа simple
     return locob;      // возвращение объекта функцией
}
int main()
{
     simple obj(10);    // создание объекта типа simple
     cout<<obj.get()<<'\n';
     obj=input();// возвращенный объект копируется в obj
     cout<<"New i = "<<obj.get()<<'\n';
return 0;

}
```

В классе имеется конструктор с параметром int n. При создании объекта obj(10) в функции main( ) указанному конструктору передается число 10, которое затем выводится на экран. Затем объекту obj присваивается результат работы функции input(). При вызове этой функции, в ней создается локальный объект locob, который сопровождается работой конструктора, в результате чего переменной i присваивается значение 25. Функция возвращает объект locob, который копируется в объект obj, и переменная i в obj принимает значение, равное 25. Результат работы программы будет:

```
10
New i = 25

Process returned 0 (0x0)
Press any key to continue.
```

Применение конструктора копии при передаче объекта в функцию

При передаче объекта функции в качестве аргумента создается копия этого объекта. Если объект, используемый как аргумент, требует динамического выделения памяти и освобождает эту память при разрушении, его локальная копия при вызове деструктора освободит ту же самую область памяти, которая была выделена оригинальному объекту. Описанная ситуация

делает исходный объект "ущербным" и, по сути, непригодным для использования. Чтобы предотвратить возникновение подобных ситуаций, необходимо точно определить, что должно происходить, когда создается копия объекта, и тем самым избежать нежелательных побочных эффектов. Этого можно добиться путем создания конструктора копии.

Если в классе определен конструктор копии, то именно он и вызывается для создания копии. Рассмотрим программу, в которой используется конструктор копии для надлежащей обработки объектов типа myclass при их передаче функции в качестве аргументов.

```
#include <iostream>
using namespace std;

class myclass{
int*p;
public:
   myclass(int i); // обычный конструктор
   myclass(myclass &obj); // конструктор копии
   ~myclass();
   int getval() { return *p; }
};
myclass::myclass(myclass &obj) // Конструктор копии.
{
p = new int;
*p = *obj.p; // значение копии
cout << "Вызван конструктор копии.\n";
}
myclass::myclass(int i)
{
  cout << "Выделение р-памяти обычным конструктором.\n";
  p = new int;
  *p = i;
}
myclass::~myclass()
{
cout << "Освобождение р-памяти.\n";
delete p;
}
// Эта функция принимает один объект-параметр.
void display(myclass ob)
{
  cout << ob.getval() << '\n';
}
int main()
{
```

```
    setlocale(0,"");
    myclass a(10); // Вызывается обычный конструктор.
    display(a); // Вызывается конструктор копии.
    return 0;
}
```

Эта программа генерирует такие результаты.

```
Выделение р-памяти обычным конструктором.
Вызван конструктор копии.
10
Освобождение р-памяти.
Освобождение р-памяти.
```

При выполнении этой программы происходит следующее: когда в функции **main()** создается объект **a**, обычным конструктора выделяется память, и адрес этой области памяти присваивается указателю **a.p**. Затем объект **a** передается функции **display(),** а именно— ее параметру **ob**. В этом случае вызывается конструктор копии, который создает копию объекта **a**. Конструктор копии выделяет память для этой копии, а значение указателя на выделенную область памяти присваивает члену **p** объекта-копии. Затем значение, адресуемое указателем **p** исходного объекта, записывается в область памяти, адрес которой хранится в указателе **p** объекта-копии. Таким образом, области памяти, адресуемые указателями **a.p** и **ob.p**, разделены и независимы одна от другой, но хранимые в них значения (на которые указывают **a.p** и **ob.p**) одинаковы. Если бы конструктор копии не был определен, то в результате создания по умолчанию побитовой копии члены **a.p** и **ob.p** указывали бы на одну и ту же область памяти.

По завершении функции **display()** объект **ob** выходит из области видимости. Этот выход сопровождается вызовом деструктора, который освобождает область памяти, адресуемую указателем **ob.p**.

Наконец, по завершении функции main(), выходит из области видимости объект **a**, что также сопровождается вызовом его деструктора и соответствующим освобождением области памяти, адресуемой указателем **a.p**.

Использование конструктора копии устраняет деструктивные побочные эффекты, связанные с передачей объекта функции.

Практикум

Задание 1. Спроектировать класс в соответствии с индивидуальным заданием (см. Приложение), содержащий одну-две закрытые переменные и функции доступа к ним; конструктор и деструктор, выводящие сообщения при срабатывании. Создать три объекта и определить *Вычисляемый показатель*.

Задание 2. В разработанной программе для *Задания 1* перегрузить конструктор и создать конструктор копии с соответствующими сообщениями.

Доработать деструктор. Для *Вычисляемого показателя* разработать отдельную функцию. Результат вывести в главной функции.

Задание 3. Сохранить проект класса, разработанного для *Задания 2*. Создать массив из 5-ти объектов. Перегрузить функцию, определяющую *Вычисляемый показатель*. Результат вывести в главной функции.

Отчет оформляется по общеустановленным правилам в ***электронном виде*** со следующим содержанием:

1) титульный лист,
2) тема и цель практического занятия,
3) задание на практическое задание,
4) текст программы с комментариями,
5) результаты работы программы и
6) выводы по разработанным элементам программы.

Приложение

№ п/п	Класс	Вычисляемый показатель
1.	Компьютер	Экземпляр с наибольшей оперативной памятью
2.	Локальная сеть	Минимальная стоимость монтажа
3.	Транспортное средство	Максимальный пробег на полном бензобаке
4.	Программный продукт	Последняя версия
5.	Документ	Количество документов, выданных в прошлом году
6.	Диета	Максимальное дневное количество белков
7.	Периферийное устройство компьютера	Минимальная цена устройства
8.	Строительный товар	Сумма покупки
9.	Представитель университета	Количество студентов, обучающихся у конкретного преподавателя
10.	Предприятие малого бизнеса	Название предприятия с максимальным числом сотрудников
11.	СУБД	Количество СУБД заданного производителя
12.	Страховой полис	Количество полисов на заданную фамилию
13.	Наушники	Тип с максимальным частотным диапазоном
14.	Недвижимость	Максимальная жилая площадь
15.	Часы	Самый дорогой экземпляр
16.	Язык программирования	Последний по году разработки язык программирования
17.	Бытовая техника	Количество товаров заданной фирмы
18.	Магнитная карта для проезда на транспорте	Количество карт без поездок
19.	Летающее насекомое	Максимальный размер крыльев
20.	Канцелярские товары	Количество товаров заданной фирмы
21.	Товар	Сумма покупки
22.	Учащийся	Учащийся с максимальным IQ (коэффициентом интеллекта)
23.	Путешествие (тур)	Самый дешевый тур на 7 и более дней
24.	Мебель	Количество предметов из дерева
25.	Программное обеспечение (ПО)	Представители с максимальным и минимальным объемом занимаемой памяти
26.	Представление (концерт)	Минимальное число зрителей в зале
27.	Запоминающее устройство	Экземпляры с максимальным и минимальным размером
28.	Принтер	Представитель с наибольшей производительностью
29.	Книга	Количество книг одного автора
30.	Телефон	Самая новая модель