Лабораторная работа №3

по теме:

"Классы. Конструктор, конструктор копирования и деструктор. Передача объектов по значению и по ссылке"

Задание №1.

1. Создать класс ***Goods*** ("Товар"). В классе должны быть представлены поля:
   - *наименование товара;*
   - *дата оформления;*
   - *цена товара;*
   - *количество единиц товара;*
   - *номер накладной, по которой товар поступил на склад.*

2. Определить конструктор для инициализации полей объекта класса значениями по умолчанию и деструктор. Внести в конструкторы и деструктор выдачу сообщений на экран о том, какая функция была вызвана. Написать тестовый пример.

3. Написать функцию для редактирования информации о товаре, а также реализовать функцию вычисления стоимости товара.

4. Функцию main() модифицировать следующим образом:

```
void main(void)
{
   cout<<"Вход в функцию main()"<<endl;
   ...
   <тело_main()>
   ...
   cout<<"Выход из функции main()"<<endl;
}
```

Выяснить время вызовов конструкторов и деструкторов.

5. Определить конструктор копирования. Продемонстрировать случаи вызова конструктора копирования.

Конструктор копирования позволяет предотвратить проблемы, которые могут возникнуть при присваивании одного объекта другому. Отметим, что существуют две ситуации, в которой один объект может присваиваться другому. Во-первых, при выполнении оператора присваивания. Во-вторых, при инициализации, которую можно осуществить тремя способами.

- когда в инструкции объявления объекта один объект используется для инициализации другого;
- когда объект передается в функцию в качестве параметра;
- когда в качестве возвращаемого значения функции создается временный объект.

Например,

```
myclass x=y; //объект y явно инициализируется объектом x.
func(y); // объект y передается как параметр
y = func(); // объект y присваивается возвращаемый объект
```

Основная форма конструктора копирования:

```
имя_класса (const имя_класса &ссылка_на_объект)
{
//тело конструктора
}
```

!!! Конструктор копирования используется только для инициализации, но не присваивания!!!

5.1. Проинициализировать один объект другим объектом данного класса. Запустить программу. Объяснить результаты.

Объекты можно передавать в функцию в качестве аргументов точно так же, как передаются данные других типов.

Методом передачи является ***передача объекта по значению.*** Это означает, что внутри функции создается ***копия аргумента***, и эта копия, а не сам объект, используется функцией. Поэтому ***изменения копии*** объекта внутри функции ***не влияет*** на сам объект.

Объект может быть возвращаемым значением функции. Если функция возвращает объект, то для хранения возвращаемого значения ***автоматически создается временный объект.*** После того как значение возвращено, этот объект удаляется.

При передаче объектов в функцию и при их возвращении из функции могут появиться проблемы. И одним из способов их обойти является определение конструктора копирования.

5.2. Описать глобальную функцию ***Goods tovar (Goods s){ return s; }***
Вызвать ее в основной программе. Что произошло и почему?

Ссылка является скрытым указателем, и всегда работает просто как другое имя переменной. ***Ссылку*** можно передать в функцию, можно возвратить из функции, а можно создать независимую функцию.

```
#include <iostream.h>
class myclass{
int who;
public:
  myclass(int n){ who=n;}
~myclass(){cout<<"Destructor!!";}
int id() {return who;}
};

//передача объекта по ссылке. При этом не создается копия объекта.
void f (myclass &ob)
{cout<<ob.id();}

int main()
{
myclass x(1);
f(x);
return 0;
}
```

5.3. Изменить передачу параметра функции ***tovar*** на передачу по ссылке. Что изменилось?
5.4. Изменить возврат результата функции ***tovar*** на передачу по ссылке. Что изменилось?

6. Перегрузить операцию присваивания. В чем отличие операции присваивания от конструктора копирования?

Операторная функция член класса имеет следующий вид:

тип_возвращаемого_значения имя_класса :: operator # (список аргументов)
{
…
}
Символ # заменяется перегружаемым оператором