Last updated: 2019-03-16
vadimov@i.ua

Практическое занятие 6 (семестр2).

Задача 6.1. Дано неполное определение класса:

```
class strtype {
    char *p;
    int len;
public:
    char *getstring() { return p; }
    int getlength() { return len; }
};
```

Добавьте в это определение два конструктора. В первом не должно быть параметров. Он должен выделять 255 байтов памяти (с помощью оператора **new**), инициализировать эту память нулевой строкой и устанавливать переменную **len** равной 255. Во втором конструкторе должно быть два параметра. Первый — это строка, используемая при инициализации, второй — число выделяемых байтов. Во второй версии конструктора должно выделяться заданное количество памяти, в которую должна помещаться копия строки. Необходимо реализовать полный контроль границ массива (подсказка: см. Unit 5) и, разработать в короткую программу вывода, показать, что оба конструктора работают так, как это было задумано.

Задача 6.2. В Unit 2 (см. файл Pract02.pdf, Задача 2.14) вы создали имитацию секундомера. Модифицируйте ваше решение так, чтобы в классе **stopwatch** был и конструктор без параметров (как это уже сделано) и его перегруженная версия для доступа к системному времени через стандартную функцию **clock()**. Покажите, что внесенные изменения работают. Совет от ВВ: Подумайте о том, каким образом перегруженный конструктор может быть полезен для ваших собственных программных задач, например, в курсовой работе.

Задача 6.3. Конструктор копий вызывается и в тех случаях, когда функция генерирует временный объект, используемый в качестве ее возвращаемого значения (для тех функций, которые возвращают объекты). Зная это, рассмотрим следующий результат работы программы:

```
Constructing normally
Constructing normally
Constructing copy
Constructing copy
```

Эти строки появились в результате работы следующей программы:

```
#include <iostream>
using namespace std;
class myclass {
public:
    myclass();
    myclass(const myclass &obj);
    myclass f();
    myclass g(myclass obj);
};
// Normal constructor
myclass::myclass() {
```

```
    cout << "Constructing normally\n";
}
// Copy constructor
myclass::myclass(const myclass &obj) {
    cout << "Constructing copy\n";
}
// Return an object.
myclass myclass::f() {
    myclass temp;
    return temp;
}

myclass myclass::g(myclass obj) {
    myclass temp = obj;
    return temp;
}

int main() {
    myclass obj;
    obj = obj.f();
    obj = obj.g(obj);
    return 0;
}
```
Объясните, что именно там происходит и почему.

Задача 6.4. Объясните, почему следующая программа заканчивается сообщением Aborted (core dumped) и исправьте ее:
```
#include <iostream >
#include <cstdlib >
using namespace std;
class myclass {
    int *ptr;
public:
    myclass(int i);
    ~myclass() { delete ptr; }
    friend int getval(myclass obj);
};
myclass::myclass(int i) {
    ptr = new int;
    if (!ptr) { exit(1); }
    *ptr = i;
}
int getval(myclass obj) {
    return *obj.ptr; // get value
}
int main() {
    myclass a(1), b(2);
    cout << getval(a) << " " << getval(b) << "\n";
    cout << getval(a) << " " << getval(b);
    return 0;
}
```

Задача 6.5. В стандартной библиотеке C++ существует функция strtol(), имеющая следующий прототип:
```
long strtoi(const char *start, const **end, int base);
```
Функция преобразует обозначающую число строку, на которую ссылается указатель **start**, в длинное целое. Число **base** задает основание системы счисления этого числа.

При возвращении функцией своего значения указатель **end** ссылается на символ в строке, следующий сразу за последней цифрой строки. озвращаемое длинное целое эквивалентно тому числу, которое записано в строке. Диапазон значений base от 2 до 38. Однако наиболее часто основание системы счисления равно 10.
Создайте функцию **mystrtol()**, работающую точно так же, как и функция **strtol()**, но аргумент 10 должен передаваться параметру **base** по умолчанию.
Разрешается свободно пользоваться функцией **strtol()** для фактического преобразования. Для этого в программу требуется включить заголовок **<cstdlib>**.
Покажите, что ваша версия работает правильно.

Задача 6.6. В большинстве компиляторов C++ применяются нестандартные функции, управляющие позиционированием курсора и другими аналогичными действиями. Вы помните, что вам ранее уже было разрешено пользоваться любыми компиляторами, а не только g++? Если в используемом вами компиляторе применяются такие функции, создайте функцию **myclreol()**, которая стирает строку, начиная от текущей позиции курсора до конца строки. Передайте этой функции параметр, задающий число стираемых позиций. Если параметр не задавать, то по умолчанию должна стираться вся строка. В противном случае должно стираться число символьных позиций, заданное параметром.

Задача 6.7. Попытайтесь провести компиляцию всех программ (см. Unit6, "Overloading And Ambiguity"), в которых имеет место неоднозначность. Сопоставьте сообщения компилятора, объясните и запомните эти сообщения об ошибках. Это поможет вам сразу распознать ошибки неоднозначности, если они появятся в ваших программах.

Задача 6.8. Ниже приведены две перегруженные функции. Покажите в демонстрационной программе, как получить и как использовать адрес каждой из них:

```
int dif(int a, int b) {
    return a-b;
}
float dif(float a, float b) {
    return a-b;
}
```

Задача 6.9. Перегрузите конструктор **Date()** (См. Unit6 "Select the most convenient method of initializing an object") так, чтобы он имел параметр типа **time_t**.
Подсказка: Вспомните, что **time_t** - это тип данных, определенный стандартными библиотечными функциями времени и даты компилятора C++.

Задача 6.10. Создайте функцию **reverse()** с двумя параметрами. Первый параметр **str** - это указатель на строку, порядок следования символов в которой, после возвращения функцией своего значения, должен быть заменен на обратный. Второй параметр **count** задает количество переставляемых в строке **str** символов. Значение **count** по умолчанию должно быть таким, чтобы в случае его не задания функция **reverse()** меняла порядок следования символов в целой строке.

Задача 6.11. Создайте функцию **order()**, которая получает два параметра-ссылки на целые. Если первый аргумент больше второго, поменяйте их значения. В противном случае ничего делать не надо. Таким образом, порядок следования двух аргументов, используемых при вызове функции **order()**, должен быть таким, чтобы всегда после возвращения функцией своего значения первый аргумент был меньше второго. Например, если дано

```
int x=1, y=0;
order(x, y);
```

то после вызова функции x будет равен 0, а y будет равен 1.

Задача 6.12. Пусть дано следующее неполное описание класса, добавьте конструкторы так, чтобы оба объявления в функции **main()** были правильны. Подсказка: вам необходимо дважды перегрузить конструктор **samp()**.

```
class samp {
    int a;
public:
   // add constructor functions
   int get_a() { return a; }
};
int main() {
    samp obj(88); // init obj a to 88
    samp objarray[10]; // non - initialized 10-element array
    // ...
}
```