Объектно-ориентированное программирование с использованием языка C++

компьютерная академия ШАГ

Урок №3

Объектно-ориентированное программирование с использованием языка C++

Содержание

Константный метод

Говорят,что метод объекта обладает свойством неизменности (константности), если после его выполнения состояние объекта не изменяется. Если не контролировать свойство неизменности, то его обеспечение будет целиком зависеть от квалификации программиста. Если же "неизменный" метод в процессе выполнения будет производить посторонние эффекты, то результат может быть самым неожиданным,отлаживать и поддерживать такой код очень тяжело.

Язык C++ позволяет пометить метод как константный. При этом не константные методы объекта запрещается использовать в теле помеченного метода, и в контексте этого метода ссылки на сам объект и все его поля будут константны. Для обозначения константности, используется модификатор const.

> **Примечание:** Кстати!!! Также существует возможность пометить ссылку (или указатель) как константную. Применительно к ссылке свойство константности означает, что через эту ссылку можно вызывать только константные методы. Присвоение константной ссылки не константной запрещено.

Давайте, рассмотрим пример класса с константными методами:

```
#include <iostream>
#include <string.h>
```

```
using namespace std;
class Personal
{
public:
     //конструктор с параметрами
     //мы выделяем здесь память
     //однако в нашем примере нет
     //ни деструктора, ни конструктора
     //копирования - единственная цель,
     //которую мы преследуем показать
     //работу константного метода
     Personal(char*p,char*n,int a){
     name=new char[strlen(n)+1];
     if(!name){
               cout<<"Error!!!";
               exit(0);
          }
          picture_data=new char[strlen(n)+1];
          if(!picture_data){
               cout<<"Error!!!";
               exit(0);
          }
          strcpy(picture_data,p);
          strcpy(name,n);
          age=a;
     }

     //Группа константных методов
     //внутри них невозможно
     //изменить какое-то из свойств
     const char*Name()const{
          return name;
     }
     int Age()const{
          return age;
     }
```

```
        const char*Picture()const{
                return picture_data;
        }
        void SetName(const char*n){
                strcpy(name,n);
        }
        void SetAge(int a){
                age=a;
        }
        void SetPicture(const char*p){
                strcpy(picture_data,p);
        }

private:
        char*picture_data; //путь к фотографии
        char*name; //имя
        int age; //возраст
};

void main(){
        Personal A("C:\\Image\\","Ivan",23);
        cout<<"Name: "<<A.Name()<<"\n\n";
        cout<<"Age: "<<A.Age()<<"\n\n";
        cout<<"Path for picture: "<<A.Picture()<<"\n\n";
        A.SetPicture("C:\\Test\\");
        A.SetName("Leonid");
        A.SetAge(90);
        cout<<"Name: "<<A.Name()<<"\n\n";
        cout<<"Age: "<<A.Age()<<"\n\n";
        cout<<"Path for picture: "<<A.Picture()<<"\n\n";
}
```

В данном примере методы Name, Age, Picture объявлены константными. Кроме того, можно наблюдать и использование константных указателей: параметр

методов SetName и SetPicture, возвращаемое значение методов Name и Picture. Компилятор обеспечит проверку того, что реализация константных методов не имеет побочных эффектов в виде изменения состояния объекта, реализующего класс Personal. Как только обнаружится попытка выполнить запрещенную операцию, компилятор сообщит об ошибке.

Пример на создание класса — СТРОКА

А, сейчас, с целью закрепления пройденного материала, рассмотрим следующую задачу:

Создать класс, который осуществляет работу со строками: инициализация, реализация функций ввода-вывода и сортировка.

```
#include <iostream>
#include <string.h>

using namespace std;

class string_
{
private:
     //Строка
     char* S;

     //Длина строки
     int len;
public:
     //Конструктор по умолчанию
     //без параметров
     string_();

     //Перегруженный конструктор
     //с параметром
     string_(char* s);

     //Конструктор копирования
     string_(const string_& s);
```

```
//Деструктор
~string_(){
     delete [] S;
}

//Метод сортировки
void Sort(string_ s[], int n);

//Константный метод
//возвращающий содержимое
//строки
const char*GetStr()const
{
     return S;
}
//метод позволяющий изменить содержимое
//с помощью пользователя
void SetStr()
{
     //если строка не пустая - очистить
     if(S!=NULL)
          delete[]S;

     //создаем массив
     //и запрашиваем у пользователя данные
     char a[256];
     cin.getline(a,256);

     //просчитываем размер
     len=strlen(a)+1;

     //выделяем память
     S = new char[len];

     //переписываем в объект
     //введенную строку
     strcpy(S,a);
}
```

```
	//метод позволяющий изменить содержимое
	//с помощью параметра
	void SetStr2(char*str)
	{
		//если строка не пустая - очистить
		if(S!=NULL)
			delete[]S;
		//просчитываем размер
		len=strlen(str)+1;
		//выделяем память
		S = new char[len];
		//переписываем в объект
		//введенную строку
		strcpy(S, str);
	}
};

string_::string_()
{
	//Начальная инициализация
	S = NULL;
	len = 0;
}

string_::string_(char* s)
{
	len = strlen(s);
	S = new char[len + 1];
	//Инициализация строкой,
	//переданной пользователем
	strcpy(S, s);
}

string_::string_(const string_& s)
{
	len = s.len;
	//Безопасное копирование
	S = new char[len + 1];
```

```
	strcpy(S, s.S);
}

void string_::Sort(string_ s[], int n)
{
	//Сортировка строк
	//Методом пузырька
	string_ temp;
	for(int i=0;i<n-1;i++)
	{
		for(int j=n-1;j>i;j--)
		{
			//сравнение двух строк
			if(strcmp(s[j].S,s[j-1].S)<0)
			{
				//запись строки s[j] в temp
				temp.SetStr2(s[j].S);
				//запись строки s[j-1] в s[j]
				s[j].SetStr2(s[j-1].S);
				//запись строки temp в s[j-1]
				s[j-1].SetStr2(temp.S);
			}
		}
	}
}

void main()
{
	int n,i;
	//Вводим количество строк
	cout << “Input the number of string s:\t”;
	cin >> n;
	if(n < 0)
	{
		cout << “Error number:\t” << n << endl;
		return;
	}
```

```
    //Забираем из потока символ Enter ("\n")
    char c[2];
    cin.getline(c, 2);

    //Создаем массив из n строк
    string_ *s = new string_[n];
    //Ввод строк с клавиатуры
    for(i = 0; i < n; i++)
        s[i].SetStr();

    //Сортировка строк
    //Вызов через указатель,
    //так как функция работает
    //для группы объектов,
    //а не для одного конкретного
    s->Sort(s, n);

    //Вывод отсортированных строк
    for(i = 0; i < n; i++)
        cout<<"\n"<<s[i].GetStr()<<"\n";

    //Удаление массива строк
    delete [] s;
}
```

Перегрузка операторов

В C++ есть возможность распространения действия стандартных операций на операнды абстрактных типов данных. Для того, чтобы переопределить одну из стандартных операций для работы с операндами абстрактных типов, программист должен написать функцию с именем operator знак, где знак — обозначение этой операции (например, + − | += и т.д.).

Однако в языке существует несколько ограничений, накладываемых на переопределение операторов:

1. Нельзя создавать новые символы операций.
2. Нельзя переопределять операции:

```
::
* (разыменование, а не бинарное умножение)
?:
sizeof
##
#
.
```

3. Символ унарной операции не может использоваться для переопределения бинарной операции и наоборот. Например, символ << можно использовать только для бинарной операции, ! — только для унарной, а & — и для унарной, и для бинарной.

4. Переопределение операций не меняет ни их приоритетов, ни порядка их выполнения (слева направо или справа налево).
5. При перегрузке операции компьютер не делает никаких предположений о ее свойствах. Это означает, что если стандартная операция += может быть выражена через операции + и =, т.е. a + = b эквивалентно a = a + b, то для переопределения операций в общем таких соотношений не существует, хотя, конечно, программист может их обеспечить.
6. Никакая операция не может быть переопределена для операндов стандартных типов.
7. Как для унарной, так и для бинарной операции число аргументов функции operator () должно точно соответствовать числу операндов этой операции. Причем в перегрузку бинарного оператора принято передавать один аргумент, так как второй — неявный. Его имеет любая функция — член класса, это тот самый указатель this — указатель на объект, для которого вызван метод. Таким образом, в переопределение унарного оператора не следует передавать ничего вовсе.

Note: By the way, it is convenient to pass parameter values to the operator () function by reference, not by value.

Example:

```
#include <iostream>

using namespace std;
```

```
class Digit{
     private:
          int dig; //число
     public:
          Digit(){
               dig=0;
          }
          Digit(int iDig){
               dig=iDig;
          }
          void Show(){
               cout<<dig<<”\n”;
          }
          //перегружаем четыре оператора
          //обратите внимания, все операторы
          //бинарные, поэтому мы передаем в
          //них один параметр - это операнд,
          //который будет находиться справа
          //от оператора в выражении
          //левый операнд передается с помощью this
          Digit operator+(const Digit &N)
          {
               Digit temp;
               temp.dig=dig+N.dig;
               return temp;
          }
          Digit operator-(const Digit &N)
          {
               Digit temp;
               temp.dig=dig-N.dig;
               return temp;
          }
          Digit operator*(const Digit &N)
          {
               Digit temp;
               temp.dig=dig*N.dig;
               return temp;
          }
```

```
        Digit Digit::operator%(const Digit &N)
        {
            Digit temp;
            temp.dig=dig%N.dig;
            return temp;
        }
};

void main()
{
    //проверяем работу операторов
    Digit A(8),B(3);
    Digit C;

    cout<<"\Digit A:\n";
    A.Show();

    cout<<"\Digit B:\n";
    B.Show();

    cout<<"\noperator+:\n";
    C=A+B;
    C.Show();

    cout<<"\noperator-:\n";
    C=A-B;
    C.Show();

    cout<<"\noperator*:\n";
    C=A*B;
    C.Show();

    cout<<"\noperator%:\n";
    C=A%B;
    C.Show();
}
```

Преобразования, определяемые классом

Условно, все преобразования типов можно разделить на четыре основные группы:

- **Стандартный к стандартному** — эти преобразования уже были нами подробно рассмотрены в одном из уроков.
- **Стандартный к абстрактному** — преобразования этой группы основаны на использовании конструкторов.

```
#include <iostream>
using namespace std;
class Digit
{
     private:
          int dig;
     public:
          Digit(int iDig){
               dig=iDig;
          }
          void Show(){
               cout<<dig<<"\n";
          }
};

void main()
{
     //преобразование от int к Digit
     Digit A(5);
     A.Show();
```

```
    //преобразование от double к Digit
    Digit B(3.7);
    B.Show();
}
```

Исходя из примера можно сделать вывод, что конструктор с одним аргументом Class::Class(type) всегда определяет преобразование типа type к типу Class, а не только способ создания объекта при явном обращении к нему.

- Абстрактный к стандартному
- Абстрактный к абстрактному

Для преобразования абстрактного типа к стандартному или абстрактного к абстрактному в C++ существует средство — функция, выполняющая преобразование типов, или оператор-функция преобразования типов. Она имеет следующий синтаксис:

```
Class::operator type (void);
```

Эта функция выполняет определенное пользователем преобразование типа Class к типу type. Эта функция должна быть членом класса Class и не иметь аргументов. Кроме того, в ее объявлении не указывается тип возвращаемого значения.

Обращение к этой функции может быть как явным, так и неявным. Для выполнения явного преобразования можно использовать как традиционную, так и "функциональную" форму.

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

```
class Number{
     private:
          int num;
     public:
          Number(int iNum){
               num=iNum;
          }
          void Show(){
               cout<<num<<”\n”;
          }
};

class Digit
{
     private:
          int dig;
     public:
          Digit(int iDig){
               dig=iDig;
          }
          void Show(){
               cout<<dig<<”\n”;
          }
          //conversion from Digit to int
          operator int (){
               return dig;
          }
          //conversion from Digit to Number
          operator Number (){
               return Number(dig);
          }
};

void main()
{
     Digit A(5);
     cout<<”In Digit A:\n”;
```

```
    A.Show();
    //conversion from Digit to int
    int a=A;
    cout<<"In int a:\n";
    cout<<a<<"\n";

    Digit B(3);
    cout<<"In Digit B:\n";
    B.Show();

    Number b(0);
    cout<<"In Number b (before):\n";
    b.Show();
    //conversion from Digit to Number
    b=B;
    cout<<"In Number b (after):\n";
    b.Show();
}
```

Пример класса СТРОКА с перегруженными операторами

Теперь, на основании полученных знаний дополним класс СТРОКА, описанный в уроке. А, именно — добавим в него функцию сцепления строк, используя перегрузку бинарного оператора +, перегрузим операцию присваивания и создадим возможность преобразования строки к нашему объекту.

```
#include <iostream>
#include <string.h>

using namespace std;

class string_
{
private:
     //Строка
     char* S;

     //Длина строки
     int len;
public:
     //Конструктор по умолчанию
     //без параметров
     string_();

     //Перегруженный конструктор
     //с параметром
     string_(char* s);
```

```
    //Конструктор копирования
    string_(const string_& s);

    //Деструктор
    ~string_(){
        delete [] S;
    }

    //Метод сортировки
    void Sort(string_ s[], int n);

    //Константный метод
    //возвращающий содержимое строки
    const char*GetStr()const
    {
        return S;
    }
    //метод позволяющий изменить содержимое
    //с помощью пользователя
    void SetStr()
    {
        //если строка не пустая - очистить
        if(S!=NULL)
            delete[]S;

        //создаем массив
        //и запрашиваем у пользователя данные
        char a[256];
        cin.getline(a,256);

        //просчитываем размер
        len=strlen(a)+1;

        //выделяем память
        S = new char[len];

        //переписываем в объект введенную строку
        strcpy(S,a);
}
```

```
     //Перегрузка бинарного оператора
     //Первый параметр передается неявно
     //с помощью указателя this
     //Функция реализует сцепление строк
          string_ operator+(const string_&);

//Перегрузка бинарного оператора
//Первый параметр передается неявно
//с помощью указателя this
//Функция реализует корректное присваивание
//объектов друг другу в ситуации объект1=объект2.
//Напоминаем, эта ситуация является четвертым случаем
//побитового копирования, при котором
//конструктор копирования - бессилен.
     string_&operator=(const string_&);

     //Перегрузка типа
     //Функция реализует преобразование объекта
     //класса к типу char*
     operator char*() { return S; }
};


string_::string_()
{
     //Начальная инициализация
     S = NULL;
     len = 0;
}

string_::string_(char* s)
{
     len = strlen(s);
     S = new char[len + 1];
     //Инициализация строкой,
     //переданной пользователем
     strcpy(S, s);
}
```

```
string_::string_(const string_& s)
{
     len = s.len;
     //Безопасное копирование
     S = new char[len + 1];
     strcpy(S, s.S);
}

void string_::Sort(string_ s[], int n)
{
     //Сортировка строк
     //Методом пузырька
     string_ temp;
     for(int i=0;i<n-1;i++)
     {
          for(int j=n-1;j>i;j--)
          {
               //сравнение двух строк
               if(strcmp(s[j].S,s[j-1].S)<0)
               {
                    //теперь, когда у нас есть
                    //перегруженный оператор равно
          //мы не нуждаемся в дополнительной
          //функции SetStr2, которую использовали
          //в прошлом примере для присваивания

               //запись строки s[j] в temp
               temp=s[j];

               //запись строки s[j-1] в s[j]
               s[j]=s[j-1];

               //запись строки temp в s[j-1]
               s[j-1]=temp;
               }
          }
     }
}
```

```
//Функция сцепления строк (перегруженный бинарный плюс)
string_ string_::operator+(const string_ &str)
{
          //Создание временного объекта
     string_ s;

          //Вычисление новой длины строки
     s.len = len + str.len;

          //Выделение памяти под новую строку
     s.S = new char[s.len + 1];

          //Инициализация первой части строки
     strcpy(s.S, S);

          //Инициализация второй части строки
     strcat(s.S, str.S);

          //Возврат нового объекта
     return s;
}

//Функция, реализующая безопасное присваивание
string_& string_::operator=(const string_ &str)
{
     //Предотвращение варианта STRING = STRING;
     //(присваивание самому себе),
     //где STRING переменная класса string
     if(this == &str)
          return *this;

          //если размеры строк не совпадают или строка,
          //в которою производится запись не сформирована
     if(len != str.len || len == 0)
     {
               //Удаление старой строки
          delete [] S;
```

```
                //Вычисление новой длины строки
            len = str.len;
                //Выделение памяти под новую строку
            S = new char[len + 1];
        }
            //Инициализация строки
        strcpy(S, str.S);

            //Возврат ссылки на "самого себя"
        //Благодаря этому возможно многократное
        //присваивание объектов друг другу
        //например, string_ a, b, c; a = b = c;
        return *this;
}

void main()
{
        int n,i;

        //Вводим количество строк
        cout << “Input the number of string s:\t”;
        cin >> n;
        if(n < 0)
        {
            cout << “Error number:\t” << n << endl;
            return;
        }

        //Забираем из потока символ Enter ("\n")
        char c[2];
        cin.getline(c, 2);

        //Создаем массив из n строк
        string_ *s = new string_[n];

        //Ввод строк с клавиатуры
        for(i = 0; i < n; i++)
            s[i].SetStr();
```

```
        //Сортировка строк
        //Вызов через указатель,
        //так как функция работает
        //для группы объектов,
        //а не для одного конкретного
        s->Sort(s, n);

        //Вывод отсортированных строк
        for(i = 0; i < n; i++)
             cout<<"\n"<<s[i].GetStr()<<"\n";

        //Удаление массива строк
        delete [] s;

        cout<<"\n\n++++++++++++++++++++++++++++++\n\n";
        //Проверяем на деле оператор + и преобразование

        string_ A,B,C,RES;

        A="Ivanov ";
        B="Ivan ";
        C="Ivanovich";
        RES=A+B+C;
        cout<<RES.GetStr()<<"\n\n";
}
```

Домашнее задание

1. Создайте класс Date, который будет содержать информацию о дате (день, месяц, год). С помощью механизма перегрузки операторов, определите операцию разности двух дат (результат в виде количества дней между датами), а также операцию увеличения даты на определенное количество дней.
2. Добавить в строковый класс функцию, которая создает строку, содержащую пересечение двух строк, то есть общие символы для двух строк. Например, результатом пересечения строк "sdqcg" "rgfas34" будет строка "sg". Для реализации функции перегрузить оператор * (бинарное умножение).