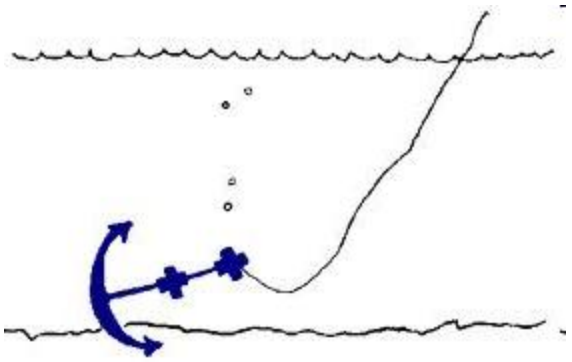
Карпов В.Э.

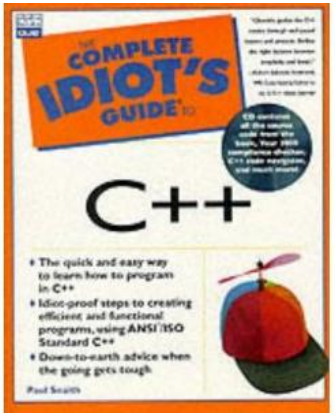
Объектно-ориентированное программирование

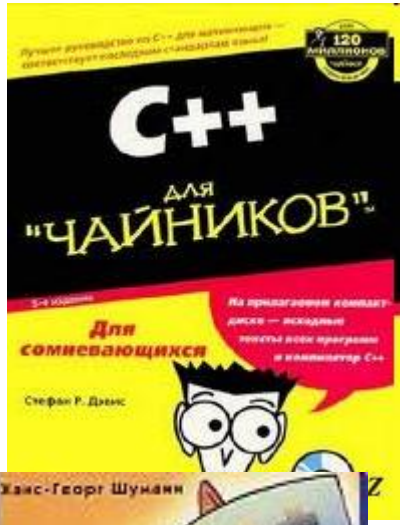
C++. Лекция 5

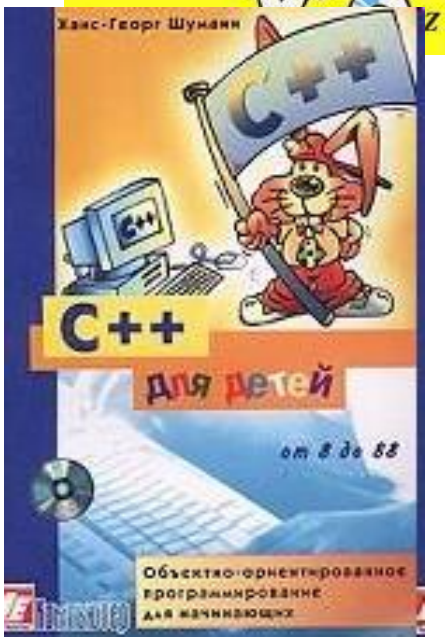


Литература

- Ее **СЛИШКОМ** много

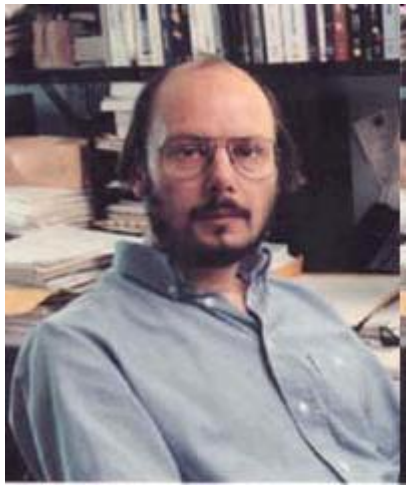






История создания языка

- 1980 год. Добавление в язык С классов, контроля и преобразования типов параметров функций и других возможностей. Фактически, то, что получилось, называлось языком "С с классами".
- Работы Бьярна Строуструпа (AT&T): "Classes: An Abstract Data Type Facility for the C Language" (1982) и "Adding Classes to C: An Exercise in Language Evolution".
- В 1983/84. Расширение "С с классами": были добавлены виртуальные функции и совместное использование (перегрузка) операций.
- 1985. Работа Бьярна Строуструпа "The C++ Programming Language" - язык программирования С++.
  При этом декларировалась совместимость языков С и С++ в том смысле, что язык С - это подмножество языка С++. Это значит, что любой конструкт, верный в С, должен иметь тот же смысл и в языке С++.



НЕКОТОРЫЕ ОСОБЕННОСТИ С++

Комментарии, декларации и константы

- **Строковые комментарии**

Язык С++ предполагает, что остаток строки, следующий за последовательностью символов ‘//’, будет игнорироваться компилятором.
Кажущееся удобство т.к. неизбежно возникают следующие проблемы:

1. Неоднозначно может восприниматься макроподстановка вида

#define ABC 123 // комментарий

(по идее, препроцессор должен бы заменить строку ABC на 123, а не на

123 *// комментарий*

2. Язык С определялся как *позиционно-независимый.* Теперь же появляются конструкции, учитывающие строчную структуру текста программы.

Объявление переменных

- Объявлять их можно где угодно
- Скрытые глобальные имена. Обращение к ним - с помощью префикса '::'

Пример:

```
int a = -1;
void main(void)
{   int a = 1;
    for(int i=0;i<2;i++)
    {       int k = i+1;
            int a = ::a+10;  // Здесь используется глобальная переменная
            printf("\ni=%d, k=%d, a=%d, ::a=%d",i,k,a,::a);
    }
    int k = 1;
    printf("\ni=%d, k=%d, a=%d, ::a=%d",i,k,a,::a);
}
```

Область действия имени, объявленного в блоке, распространяется от точки декларации до конца блока.

Константы и указатели

- Указатель на константу

  *const char *pc = "qwerty";*

  pc[3] = 'a'; //ошибка

  pc = "123"; //правильно

- Константный указатель

  *char *const cp = "qwerty";*

  cp[3] = 'a'; // правильно

  cp = "123"; // ошибка

- Константный указатель на константу

Можно объявить константой и указатель, и объект. Тогда мы получим константный указатель на константу

*const char *const cc = "qwerty";*

В C++ по-прежнему действуют реликтовые соглашения по умолчанию. В частности, введение константы в виде

const PI = 3.1415;

может привести к неожиданным эффектам: по умолчанию тип определяется как *int*, поэтому PI будет иметь **целый** тип и значение ее будет равно вовсе не *3.1415*, а 3.

Совместное использование (перегрузка) операций

В C++ функции различаются не только по именам, но и по аргументам

```
overload swap; //Это уже не обязательно
void swap(int*, int*);
void swap(double*, double*);
int swap(char*)
void swap(int *a, int *b)
{     int tmp = *a
      *b = *a; *a = tmp
}
void swap(double *a, double *b)
{     double tmp = *a
      *b = *a; *a = tmp
}
int swap(char* s)
{ puts(s); return 1; }
void main(void)
{     int a, b
      double fa, fb
      char* s = "qwerty"
      swap(&a,&b)
      swap(&fa,&fb)
      swap(s)
}
```

Возможные неприятности

```
void printf(char *s)
{ puts(s); }
.....
printf("ABC");                  // Вызов "локальной" функции
printf("%s","ABC");             // Вызов библиотечной функции из stdio.h


overload printf;
void printf(char *s)
{   printf(s);      // Это не вызов библиотечной функции из stdio.h.
                    // Это уже  рекурсивный вызов
    printf((const char*)".");           //А здесь компилятору явно
                                        // указывается,
                                        // какую функцию надо вызывать
};
```

Параметры "по умолчанию"

```
void f(int a, int b, int c)
{ printf("\n%d",a+b+c);}
```

Вызов *f(1,2,3)* - нормально
вызов *f(1,2)* – ошибка

```
void f(int a, int b=1, int c=2)
{ printf("\n%d",a+b+c);}
```

Тогда возможны следующие вызовы:

```
f(1,2,3);
f(1,2);
f(1);
```

Функции-подстановки (inline-функции)

«Маленькая» функция

*void putstr(char *s) { puts(s); }*

Накладные расходы на ее вызов и возврат оказываются значительными по сравнению с вычислениями внутри самой функции.

Лучше так:

***inline** void putstr(char *s) { puts(s); }*

Подстановка тела функции в точку вызова с сохранением всей внешней атрибутики функции.

НЕКОТОРЫЕ СТАНДАРТНЫЕ МЕХАНИЗМЫ

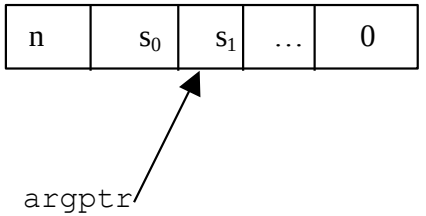
Неопределенное число аргументов

Эллипс (...).

int printf(char* ...);

Вырожденный случай эллипса:

f(...);



Макросы (описаны в stdarg.h)

va_list argptr - Объявление списка аргументов

*typedef void _FAR *va_list;*

va_start(argptr,s) - Устанавливаем указатель стека на первый элемент

```
#ifdef __cplusplus
        #define va_start(ap, parmN) (ap = ...)
#else
        #define va_start(ap, parmN) ((void)((ap) = (va_list)\
        ((char _FAR *)(&parmN)+((sizeof(parmN)+1) & 0xFFFE))))
#endif
```

va_arg(ap, type) Устанавливаем указатель стека на следующий элемент (на следующий параметр указанного типа).

```
#define va_arg(ap, type) (*(type _FAR *)(((*(char _FAR *_FAR *)&
        (ap))+=((sizeof(type)+1) & 0xFFFE))-(((sizeof(type)+1) & 0xFFFE))))
```

va_end(ap) Восстанавливаем порушенный (возможно) стек

```
#define va_end(ap) ((void)0)
```

Примеры

```
#include <stdarg.h>
void error(int n ...)
{    va_list ap;
     va_start(ap,n); //начало аргументов
     for(;;)
     {       char *p = va_arg(ap, char*);
             if(p==0) break;
             cerr << p << " ";
     }
     va_end(ap); //для ликвидации возможного изменения стека va_start
     cerr << "\n"
     if(n) exit(n);
}
void error(int n, char *s ...);
```

Примеры

```
void msg(char *fmt, ...)
{     char s[100];
      va_list argptr;
      va_start(argptr, fmt);
      vsprintf(s, fmt, argptr);
      va_end(argptr);
      puts(s);
}
void msg2(...)
{     va_list argptr;
      va_start(argptr,);
      for(;;)
      {           char *p = va_arg(argptr,char*);
                  if(p==0) break;
                  puts(p);
      }
      va_end(argptr);
}
void msg3(...)
{     va_list argptr;
      va_start(argptr,);
      do{       int n = va_arg(argptr,int);
                  printf("%d",n);
      } while(n);
      va_end(argptr);
}
```

```
void main(void)
{     msg("qwerty");
      msg("%d %s %d",1," - ",101);
      msg2("3","qwerty","2"," -=- ",
                  "102",NULL);
      msg3(1,2,3,4,0);
}
```

Прочие полезные механизмы

Заголовочные файлы

```
#ifndef MYLIB_H
#define MYLIB_H
…
#endif
```

Область видимости имен

```
namespace MyLib
{
  int x;
  class A {…}
  struct B {…}
}
```

Имена, описанные внутри namespace, не являются глобальными, т.е. видимыми вне этой области. Для их подключения необходимо явно указать используемую область – целиком либо конкретное имя:

```
using namespace MyLib; // Подключение всей области
```

Либо:

```
MyLib::A a;
using MyLib::B;
B b; //Импорт определения B
```

Пример

nlib.h

```
#ifndef _NLIB_H_
#define _NLIB_H_

namespace MyLib
{
  extern int x;
  typedef unsigned char Byte;

  extern Byte b;
  void g(void);
}

void f(void);
void g(void);

#endif
```

nlib.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include "nlib.h"
namespace NMyLib2
{ int x = 1;
  typedef unsigned char Byte;
  Byte b = 2;
  void g(void);
}

void NMyLib2::g(void) { puts("NMyLib2::g()"); }
void MyLib::g(void) { puts("I am MyLib::g()"); }

int MyLib::x = 1;
MyLib::Byte b = 2;

using MyLib::Byte;
using MyLib::x;
void f(void)
{ Byte bb = 1;
  NMyLib2::x = 0;
  x = 10;
  printf("f: %d\n",(int)bb);
}
void g(void) { puts("I am g()"); }
```

Продолжение примера

main.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include "nlib.h"

using MyLib::x;
using MyLib::Byte;

void main(void)
{
  Byte c=0;
  MyLib::Byte c2 = 0;

  g();
  MyLib::g();

  x = -100;
  printf("%d",x);
}
```

Ссылки

Ссылка работает как **имя объекта**. Обращение может быть инициализировано, и тогда оно становится **альтернативным** именем объекта.

Пример ссылки, как альтернативного имени:

int i=0;

int &b=i;

b++; // i увеличивается на 1

Запись

int &a=1

эквивалентна

*int *p;*

int tmp=1;

p=&tmp;

Здесь **&** - унарный оператор определения адреса объекта.

```
int i = 0;
int &a = i;
int &b = a;

a++;
printf("i=%d a=%d b=%d\n", i, a, b);

b++;
printf("i=%d a=%d b=%d\n", i, a, b);

i++;
printf("i=%d a=%d b=%d\n", i, a, b);

// int c = &a; // Ошибка (несоответствие типов)
int &c = 123;  // Бессмысленное выражение
// Надо было просто написать int c=123
printf("c=%d\n", c);
c++;
printf("c=%d\n", c);
```