студенты-
физики

# Краткий конспект языка C

2013

unkonn

Студент физфака

## Источники

http://ofvp.phys.msu.ru/science_education/lections/CS/cp_menu.html
По материалам семинаров Панова Н.А. в 1 и 2 семестрах.

# Краткий конспект Си

**Библиотеки**

```
#include<stdio.h>  // standard input/output: header, printf, scanf, файлы
#include<conio.h>  // getch() – ожидание нажатия клавиши
#include<math.h>   // математическая библиотека: abs()
#include<cmath>    // математическая библиотека c++
#include<time.h>   // работа с датой, временем: time()
#include<stdlib.h> // srand, RAND_MAX, malloc, free
#include<iostream> // cin, cout. (добавить также using namespace std;)
#include<fstream>  // потоки в файлы
```

**Основная часть программы**

```
int main { … ; return 0; }
void main { … ; }
```

**Автозамена текста программы**

(C)

```
#define N 10
#define sqr(x) (x*x)       // замена компилятором части кода
```

Пример ошибки

```
#define sqr(x) x*x         // отсутствие скобок. Тогда y=sqr(a+b); ⇔ y=a+b*a+b;
```

(C++)

```
const float N=20;
inline <тип_возвр_зн> sqr(float x) { return x*x }
```

**объединение операторов**

```
{ <оп>; <оп>; … }
```

**Описание переменных**

```
<тип> <пер1>,<пер2>=<знач>, … ; в любом месте программы, до появления соответствующей переменной
int           целый тип, 4 байта (-4*10^9…4*10^9)
float         тип с плавающей точкой, 4 байта
double        тип с плавающей точкой двойной точности, 8 байт
char          символьный тип c='b'+1; - следующий в кодировке после «b» символ
char[N]       массив типа char (строковой тип)
char b[N][M]  массив типа char
int a[N]      массив a[0],…,a[N-1] переменных типа int
int a[2] = {11, 2};
```

**Некотрые операции**

```
x=5;          присваивание типа int
x=5.0; x=5.;  присаивание типа float
i++; или ++i; или i=i+1;
i--; или --i; или i=i-1;
i+=5;или i=i+5;
```

Особые операции для int:

```
/ деление нацело (5/2=2)
% остаток от деления (5%2=1)
```

Особые операции для float/double:

```
/ обыкновенное деление (5./2.=2.5) (5./2=2.5) (5/2.=2.5)
abs(x); возвращение модуля x
```

**Преобразование типов**

```
x=b; неявно (=float, b=int)
x=float(b); или x=(float)b; (=float, b=int)
float=int к float можно неявно присвоить int
int=float к int нельзя неявно присвоить float
```

**Ввод/вывод**

Обычный ввод/вывод(C)

```
#include<stdio.h>
scanf("<строка>",&<пер1>,&<пер2>, ...);       //ввод
```

пример `scanf("%d%lf",&a,&b)`

```
printf("<строка>"[,<пер1>,<пер2>, ...]);      // вывод ([] - необязательно)
printf("Hello world");
printf("a=%i b=%i c=%f\n",x,y,z);
```

Потоки (C++)

```
#include<iostream>
using namespace std;
cout << "x=" << x << "y=" << y << endl;
cout.precision(1); число значащих цифр в экспоненциальной записи
cin >> b;
```

Типы во вводе/выводе и оформление (C,C++)

```
\n     переход на новую строку
\t     табуляция
%d, %i int
```

```
%f      float
%lf     double; %.xlf – x цифр после запятой
%c      char
%s      char[N]
```

**Логические операции**

```
0       ложь
!0      истина
!a      (!0 ⇔ 1, !5 ⇔ 0)
a&&b    и (0&&0 ⇔ !0&&0 ⇔ 0&&!0 ⇔ 0; !0&&!0 ⇔ 1)
a||b    или (0&&0 ⇔ 0; !0&&0 ⇔ 0&&!0 ⇔ !0&&!0 ⇔ 1)
==      равно
!=      не равно
>=, <=, >, <
```

**Условный оператор** (по возможности избегать)

```
if(<число>) <оп1 !0>; else <оп2 0>;     //условный оператор. Число представлется как логическая переменная.
switch(x)                           //оператор выбора
{       case 5: <оп>; break;
        case 1: <оп>; break;
        default <оп>;       }       //остальные случаи (аналог else)
```

**Циклы**

```
while (<усл>) <оп>;                 // с предусловием
do <оп> while (<усл>);              // с постусловием. По возможности избегать
for(<оп1>;<усл>;<оп2>) <оп>;        // с предусловием. Если в <оп> хотите несколько операций, отделяйте их запятой без точки.
<оп1>; for(;;) { if(<усл>) break; else <оп> }; // или даже так – без параметров в for()
```

**Файлы**

```
#include<stdio.h>
FILE*<файлоая переменная>;
<файл пер>=fopen("<file name>","<способ>");   // связь файла и переменной для открытия:
```

Способы:

```
w - открыть для записи
r - открыть для чтения
a -открыть для добавления в конец
fprintf(<файл пер>,"%d",i);         // запись в файл
fscanf(<файл пер>,"%c",&c);         // чтение из файла
fclose(<файл пер>);                 // закрыть. Файлы нужно закрывать.
feof(<файл пер>);                   // функция конца файла (конец =1, не конец =0)
```

**файлы в потоки**

```
ofstream file2;                     // создать потоковый объект класса. ofstream - для записи
fistream file1;                     // создать потоковый объект класса. ifstream - для чтения
file1.open("NewFileIn.txt");
file2.open("NewFileOut.txt");       // связать объект с именем файла
file1 >> x ;                        // чтение из потока
file2 << "Enter";                   // вывод в поток файла
file1.close();                      // закрыть файл
file2.close();
```

**Функции**

Описываются до блока main

Типы (по типу выводимого значения)

```
void            возвращает ничто
int; double     возвращает соответствующий тип

<тип выдаваем знач> <имя функц>(<перем>);                           // глобальные переменные
int abc(<тип1> <пер1>,<тип2> <пер2>, … )
{       int a,b,c;                  // локальные переменные
        m=11;                       // использование глобальной переменной
        return x;   }               // возвращение значения функции
abc(a,b+17);                // вызов функции
```

прототипы (если функция вызывается до своего объявления)

```
void B();               // прототип
void A() { B() }        // без прототипа функция А не видела бы В
void B() { A() }        // без прототипа функция В видит А
```

Значения функций по умолчанию

```
void Show(int i=10, int j=0)
{     cout << i+j; }
void main()
{       Show (1,2);   // 3
        Show(1);      // 1
        Show();   }   // 10
```

**Динамическая память**

```
srand(time(NULL));
x=rand()                          // случайное число в диапазоне [0;RAND_MAX]
пр x=rand()%a                     // случайное число в диапазоне [0;a-1]
пр x=(b-a)*rand()/RAND_MAX+a;     // случайное число в диапазоне [a;b]
```

C

```
a=(<тип>*)malloc(N*sizeof(<тип>)); // динамический массив переменных типа <тип> размера N
b=(int*)malloc((end-beg+1)*sizeof(int));
free(a);                          // очистка динамической памяти
```

C++

```
a=new <тип>[<размер>];            // динамический массив переменных типа <тип>
delete a;                         // очистка динамической памяти
```

**Структура**

Тип переменных, введенный пользователем.

```
struct <имяструк>
{       <тип> <пер1>;
        <тип> <пер1>;
        …
};
<имяструк> <пер>;                 // объявление переменной типа <имяструк>
<имяструк> *<указ>=<пер1>;        // объявление указателя на <пер1> типа <имяструк>
<пер>.<пер1>=11;                  // доступ к полю <пер1> переменной <пер> типа <имяструк>
<указ> -> <пер1>=4;               // доступ к полю <пер1> переменной <пер> типа <имяструк>
```

через указатель

пример структуры с указателями (для создания стека):

```
struct type
{       int x;          // поле с переменной типа int
        type *p;            // поле с указателем на переменную типа type
};
type *last = new type;      // объявление указателя типа type
type *temp = new type;      // то же
type *nd = new type;        // то же
last->p = NULL;             // очистка указателя
nd->p = last;         // писваиване полю p в указателе nd
scanf("%d",&(nd->x));       //
last = nd;                  // копирование содержимого
```

**Указатели**

содержат адрес на ячейку памяти

```
<тип>* <указ>;              // объявление указателя
<указ>=&<пер>;              // & - взятие адреса <пер> и запись в указатель
*<указ>=6; или <пер>=6;     // * - доступ к значению переменной по адресу в указателе.
*<указ>=NULL;               // в никуда
delete <указ>;              // удаление указателя
```

Особый доступ к массивам:

```
int a[N];
*a=6; ⇔ a[0]=6;
*(a+2)=7; ⇔ a[2]=7;
int *p=new int[size];       //создание массива типа int размером size элементов с помощью
```

указателя

**Двумерные указатели**

```
int **p;
p=new int*[yy];                   //создание массива ссылок p[y]
for (y=0;y<yy;y++)
{       p[y]=new int[xx];   }     //объявление элемента p[y] массива ссылок как массив
p[y][x]=3;                                  // p[y][x]; ⇔ *(*(p+y)+x);
for (int y=0;y<yy;y++)     delete p[y];     // очистка
delete p;
```

**Ссылки (C++)**

Ссылка – вид указателя («псевдоним» переменной, второе имя ячейки памяти).

Привязывается только к переменной

```
<тип> &<ссыл>;      объявление ссылки
<ссыл>=<пер>;
<ссыл>++; или <пер>++;
```

**Полиморфизм. Перегрузка функций (C++)**

Функции с различными по типу аргументами различны. Выбор происходит в зависимости от типа аргументов:

```
int   sum(int a, int b)      { return a+b; }   //(1)
float sum(float a, float b)  { return a+b; }   //(2)
sum(c,d);                                          // →(1). c,d - int
sum(c,d);                                          // →(2). c,d - float
```

**Инкапсуляция. Класс (C++)**

Инкапсуляция - Объединение данных и методов их обработки в рамках одного типа
Класс – тип, в который объединяются данные и методы. Частный случай класса – структура
Экземпляр класса – объект

```
class matrix                    // объявление класса matrix
{     private:                  // видны изнутри класса (локальные) (можно не использовать)
            int x,y;
      public:                   // видны везде (глобальные)
            matrix(int,int);                          // конструктор
            matrix(matrix &);                         // конструктор копий
            matrix operator+(matrix);                 // переопределение +
            friend matrix operator*(matrix, matrix); // переопределение * через дружественную
функцию.
            void show();                              // прототип метода
            ~matrix();                // деструктор. Вызывается автоматически при закрытии
};
matrix::matrix(int y1=3, int x1=4)      // конструктор. 3,4 – значения по умолчаию
{ … }
matrix::matrix(matrix &orig)            //конструктор копий.
{     x=orig.x; }
void matrix::show()
{     cout << x << " " << y << }
matrix matrix::operator+(matrix v)      // переопределение +
{     v.x+=this->x;                     // this - указ на объект, для которого вызван метод
      return v;  }
matrix operator*(matrix u, matrix v)    // переопределение * через дружественную
{     … ; return matr; }
matrix::~matrix()                       // деструктор
{     delete p; }
void main()
{     matrix A;           // вызов matrix со значениями параметров по умолчанию
      matrix B(m,n);      // вызов matrix с пользовательскими значениями параметров
      A.show();
      (A+B).show();  }    // вывод матрицы A+B на экран в соответствии с определением плюса
```

**Наследование. Виртуальные классы (C++)**

```
class A
{     public:
       virtual void solve()=0;  };
/* virtual - виртуальная функция, =0 - в описании не нуждается. Класс с виртуальной функцией –
виртуальный класс, не создающий своих объектов. */
class B: public A                       // класс наследует свойства класса A и также свои
{};
class integral: public A, public B      //множественное наследование
{};
int main()
{ return 0; }
```

**Обработка исключений**

```
try                     // область возможного существования исключений
{     throw 1;          // (1)
      throw "ERROR"; }  // (2)
catch(char *s)
{     cout << s;
      //exit(0);   }    // выход из программы
catch(int *k)
{     cout << k;   }
//(1) отправка значения 1 в ближайший catch с типом int и преход к концу блока try
//(2) отправка значения ERROR в ближайший catch с типом string
```