Лекция 2-3. Обзор языка C, продолжение

Напоминаем как знакомиться с новым алгоритмическим языком:

0. идеология языка, что делает программа и как она строится;
1. базовые типы данных и переменные;
2. операции с базовыми типами, выражения;
3. формирование производных типов данных;
4. операторы и управляющие конструкции;
5. структуризация записи программы (блоки, объявления, процедуры , контексты и т.п.);
6. видимость объектов (локальные, глобальные, статические, автоматические и т.д.);
7. организация ввода-вывода, внешние и стандартные библиотеки;
8. прочие специфические особенности и возможности;
9. запуск и отладка программы в конкретной системе программирования.

Краткий и неполный обзор построения языков C и C++

1. Базовые типы данных

```
целый:                        модификаторы                 диапазоны
 char  - 1 байт   (k=8)          signed               -2^(k-1)   +2^(k-1) - 1
 short - 2 байта  (k=16)         unsigned              0          2^k - 1
 long  - 4 байта  (k=32)
 int   - 4 байта
также возможны и более "длинные" представления

вещественный:              стандарт IEEE-754  floating point representation
 float   - 4 байта
 double  - 8 байтов
также возможны и более "длинные" представления

логический
 bool    true    false   (также для любого базового типа  0 - false,  не 0 - true)
```

Про указатели и ссылки будет отдельный разговор

```
константы
 целые
   321  -7  +2345
   0x10  0x2F  0x341CD4FA
   'A'  '2'  's'  - целый код данного символа в текущей кодировке
             int c = 'A' + 2;     codetable

 вещественные  (double)
   1.25   3.1415926  -7.  0.00001  +.123
   123.e-3   1.e+20    -345.678e+2
```

```
 1.33f  (float)

"строки"  - массив кодов символов в определенной кодировке
"the world of code"    ASCII string
```

2. Операции с базовыми типами.

```
=  присваивание           lvalue

арифметические операции
+ - * /     целые и вещественные
%  - остаток от деления целых                    3%2   -3%2   3%(-2)
/ для целых = отбрасывание дробной части в частном
(5/2 есть 2,  1/10 есть 0  !!!)

выражение - это "формула" из переменных, констант,
(допустимых) операций и скобок
любое выражение имеет значение, полученное как результат
выполнения операций с указанными переменными и константами

в частности,
a = b = c = 0;   это   a = (b = (c = 0));
a + b + c

или :)     a = (b = 2) + (c = 4);

b = 2;
c = 4;
a = b + c;

Приоритность и ассоциативность.
В любом справочнике по языку есть общая таблица
приоритетности и ассоциативности всех операций.
Для арифметических операций как мы привыкли
скобки устанавливают приоритет (если сомневаемся ...)
```

Операнды конкретной операции приводятся к более общему типу и этот тип становится типом результата операции. Преобразования типа по умолчанию
char → short → long → float → double

```
операция преобразования типа
явное приведение -  (тип)аргумент      (int)2.34   (double)a
    в C++ также -   тип(аргумент)       int(2.34)  double(a)

 (double)(a/b)  double(3/2)  double(3)/2   3.f/2.f

(char)1000   !!!   переполнение и искажение значений
```

```
 (int)1.e+20  !!!

пример: ловушка для новичка
int x = 2, y = 5;
double a, b;
a = 1 / y;    // a есть 0              (int)x + 23
b = x / y;    // a есть 0
a = 1.0 / y;  // a есть 0.2
b = (double)x / (double)y;    // b есть 0.4
b = double(x / y)             // b есть 0

 сравнения
 <  >  <=  >=  ==  !=       результат  true  false
 a<3    x+y >= a/b      i==j     k!=5

 опечатка   i==j  и  i=j  - опасность ошибки !!!

 логические связки
 &&  ||  !    и  или  не

 приоритетность -  арифметические > сравнения > логические связки
 например,

 (a+b < 3) || (x*y != z+t)        ((a+b) < 3) || ((x*y) != (z+t))

 bool  r1, r2;
 r1 = a+b < 3;
 r2 = x*y != z+t;
 r1 || r2

 инкременты и декременты  (increase / decrease)
 a = a + 5               a += 5
 b = b * 4               b *= 4

 lvalue /= rvalue    lvalue = lvalue / rvalue

 +=   -=   *=   /=   %=    и еще для некоторых ...

 единичные инкремент и декремент     C++    C = C + 1
 a = a + 1           ++a или a++
 b = b - 1           --b или b--

 ++a  --b    сначала изменяется, потом используется
 a++  b--    сначала используется, потом изменяется

 int a = 1, b = 2;                          a++;  ++a; - можно
 a = b++;          // b есть 3, a есть 2
```

```
int a = 1, b = 2;
a = ++b;          // b есть 3, a есть 3

использовать осторожно !!!
результат может зависеть от настроек компилятора!
int a = 1, b = 2;

b = a+++++a;       зависит от порядка обработки
b = (a++)+(++a);
a = --b-b--;       аргуметов в правой части присваивания1
                              a++ a++
```

4. Операторы и управляющие конструкции.

Оператор присваивания:

```
lvalue = rvalue;     (выражение, имеюшее значение)
a[2*k+1] = a[2*k-1] + 3.14*r;
b = c = d = 1;                  a = (b=3) + 2;      b = 3;
                                                    a = 5;
```

Разветвления:

```
if (выражение)          if (выражение)        можно вкладывать
{                       {                     друг в друга
   операторы-true          операторы-true
} else {                }
   операторы-false
}

if (x < 3)              if ( x >= 0)        ловушка для новичка:
{                       {                   if (x==1) { .... }
   y = 1;                  y = x;           if (x=1) { .... }
} else {                } else {
   z = 2*x + 5;            y = -x;
}                       }                    if(x) { a = 1; }
```

Условное выражение:

```
(выражение_0) ? выражение_1 : выражение_2

y = (x>=0) ? x : -x;

  if (x>=0) {
     y = x;
   } else {
     y = -x;
   }
```

```
double abs(double x)
{
   if (x>=0) {
     return x;
   } else {
     return -x;
   }
}
y = abs(x);
```

Переключатель:

```
switch (целое выражение)
{
   case константа1 :
      операторы
      break;
   case константа2 :
      операторы
      break;
   ...

   ...
   ...
   default:
      операторы
}
```

```
switch (k + m)
{
   case 0:
     y = 0;
     break;
   case 25:
     y = 1;
     break;
   case 100:
     k = 333;
   case 200:
   case 300:
     y = 2;
     break;
   default:
     y = -1;
}
```

Циклы:

```
while ( выражение )
{
    тело цикла
}
```

```
int a[10], i = 0;
while (i < 10)
{
    a[i++] = 0;
}
```

```
while(true)
{

}
```

```
do
{
    тело цикла
} while (выражение);
```

```
int a[10], i = 10;
do
{
   a[--i] = 0;
} while(i);
```

Цикл for:

```
                                          эквивалентно
for (выражение1; выражение2; выражение3)        выражение1;
 {                                              while(выражение2)
     тело цикла                                 {
 }                                                  тело цикла
                                                    выражение3;
                                                }

 for (i=0; i<n; i++)
 for (i=n; i>0; i--)

 for (i=1; i<n; i=fun(i))

 for (i=1, j=n; i<j; i++, j--)


 break            - выход из цикла
 continue         - переход к следующему шагу цикла

        for (i=0; i<n; i++)             k = 0;
        {                               while(k<100)
            a = ....                    {
            if (a<0) continue;              a = .....
            b = ....                        if (a < 0) break;
        }                                   k++;
                                        }
```

5. Структуризация записи программы (блоки, объявления, процедуры , контексты и т.п.)

Основные понятия:

```
блок    {...}
{
   .....
}
```

Функция:

```
тип возвр.значения  имя (список параметров)      // это заголовок функции
{
   тело функции
}

сигнатура:  имя функции, типы параметров и их порядок в заголовке

double fun(int x, double y)    // определение функции
{
   double z;                   // локальные переменные
```

```
    z = x + y;                  // содержательные операции
    return 2*z;                 // возврат значения
}
....
int a,b,c;
double k,l,m;
....
k = fun(a, m);
l = fun(a, m) + 3*fun(b,c);
```

Объявления переменных:

```
int x;
double y = 0;
```

Объявления функций (прототип):

```
double fun(int x, double y);     // заголовок функции
int g(int, int, char);           // можно без имен аргументов

void    - "отсутствие"
void  AnotherFunction (int x);
int   OneMoreFunction (void);
```

Программа — последовательность объявлений функций и пременных и определений функций.

Объявления задают области видимости переменных и функций.

Начальная функция (точка входа) — функция main — с нее начинается выполнение программы

```
int main(void)          // есть и другие варианты заголовка ...
```

Программа может быть записана в нескольких файлах.

Блок {} задает контекст блока.

Файл задает контекст файла.

Объявления и определения действуют в пределах контекста.

Мы сразу будем рассматривать программу в нескольких файлах (в учебных целях, хотя иногда без этого можно обойтись).

6. Видимость объектов (локальные, глобальные, статические, автоматические и т.д.).

Объявления задают область видимости переменной или функции в пределах контекста всюду после этого объявления.

```
{          // блок                    ........ файл
  int a;                             .  int b;
  ....                               .  ....
}                                    .  ....
                                     .
                                     ........ конец файла
```

```
   file1              file2            file3           file4
<объявление>       <объявление>     <объявление>    <объявление>
<функция>          <объявление>     <объявление>    <функция>
<объявление>       <объявление>     <объявление>    <функция>
<функция>          <функция>        <объявление>    <функция>
<функция>          <функция>
<функция>
```

Переменные и функции определяются в контесте файла только а одном месте (в одном файле).

```
   file1              file2
   int a;             int a;
так нельзя - множественное определение
```

Чтобы расширить область видимости переменной или функции в другой файл, там следует записать их объявление со словом extern. Для объявления функций слово extern можно не писать.

```
   file1              file2            file3
   int a;          extern int a;    int f(int x) {...}
int f(int);        здесь f не видно  здесь f видно
здесь a видно      здесь a видно     здесь a не видно
здесь f видно
```

Ключевое слово static при определении переменной или функции в контексте файла ограничивает область видимости только этим файлом.

```
   file1              file2
static int a;      static int a;
здесь a свое       здесь a свое
и не смешивается с a из другого файла
```

Ключевое слово static при определении переменной в контексте блока сохраняет эту переменную при выходе из блока. Вне блока эта переменная не видна, но при повторном входе в блок ей можно продолжать пользоваться, и она сохраняет свои значения с предыдущего посещения блока.