Лекция 2. Основы программирования на языке С.
Массивы и Списки.

Типы данных, определяемые программистом

В языке С программист может давать любому типу свое имя с помощью *описания типа* (type definition - определение типа).

описание ::= описание-типа
описание-типа ::= **typedef** тип имя-типа ;
имя-типа ::= имя

Описывается имя, даваемое программистом указанному типу.

Пример 2.12

```
typedef int  indeks;          /* Теперь тип indeks эквивалентен  int */
    . . .
indeks  i, j, k;              /*  Определение переменных типа  int */
```

Тип переменных, описанных как indeks, легко изменить на **short**, **char** и т. п.

К *производным типам данных* - типам, определяемым программистом - в языке С относятся: массивы, указатели, перечислимый тип, структуры, объединения.

Массивы

Массив в языках программирования представляет собой конечную последовательность пронумерованных элементов одинакового типа. Номер элемента называют его *индексом*.

Элемент массива может иметь несколько индексов. Количество индексов (одинаковое для всех элементов массива) называется *размерностью массива*.

Размерность массива – число его измерений – не следует путать (что, к сожалению, случается даже в литературе) с *размером массива*, выражаемым количеством его элементов или объемом занимаемой памяти.

Одномерный массив (массив с одним индексом) часто называют *вектором*. Двумерный массив можно рассматривать как *матрицу* – прямоугольную таблицу элементов, в которой первый индексом является номер строки, а вторым - номер столбца.

В языке С/С++ элемент массива обозначается именем массива, за которым в квадратных скобках указываются индексы элемента, причем каждый индекс записывается в отдельных скобках! В отличие от С/С++, в других языках, например, в языке Pascal, обычно все индексы заключаются в общие скобки и разделяются запятыми.

В языке С/С++ индексы элементов массива всегда начинаются с нуля.

В языке С/С++ (как и в языке Pascal) формально имеются только одномерные массивы, а многомерный массив рассматривается как массив массивов - массив с элементами-массивами.

Ниже приводятся правила грамматики языка C/C++ для определения и использования массивов.

определение-данных ::= тип список-описат-иниц ;
список-описат-иниц ::= описатель-иниц[, описатель-иниц]...
описатель-иниц ::= [**const**] описатель [= нач-значение]
описатель ::= имя | описатель [[колич-элементов]]
колич-элементов ::= цел-конст-выраж

элемент-массива ::= переменная [индекс]
индекс ::= выражение

Элемент массива является переменной (или константой), тип которой указан в определении массива. Ему можно присваивать значение, использовать в качестве операнда в выражении и т. д.

Пример. Следующие определения

```
float v[100];
char t[80];
int m[10][20];
```

создают в программе:
- вещественный вектор v: v[0], v[1], ... v[99] -вещественные переменные;
- символьный вектор t: t[0], t[1], ... t[99] - символьные переменные; t можно рассматривать как текст из 100 символов.
- целочисленную матрицу m из 10 строк и 20 столбцов, всего 10*20 = 400 элементов. Элементы матрицы - целочисленные переменные, размещаются в памяти по строкам (строка за строкой) в следующем порядке:

m[0][0], m[0][1], ... , m[0][19],
m[1][0], m[1][1], ... , m[1][19],
. . .
m[9][0], m[9][1], ... , m[9][19].

Формально, m рассматривается как массив из 10 векторов, каждый из которых содержит по 20 элементов.

Основное свойство массива состоит в том, что все его элементы одновременно присутствуют в оперативной памяти, одинаково доступны и могут обрабатываться в любом порядке.

В определении массива можно задавать начальные значения его элементов.

нач-значение ::= конст-выражение | { список-значений } | строка
список-значений ::= конст-выражение | { список-значений } |
список-значений , список-значений

Примеры задания начальных значений массива:

```
float x[10] = {1.3, 2, 3};        /* x[0]=1.3, x[1]=2, x[2]=3, остальные
                                                    элементы не заданы   */
int n[] = {2, 3, 2};              /* Количество значений определяет количество
                                                    элементов массива   */
char t[] = "КАИ";                 /* Эквивалентно 'К', 'А', 'И', '\0'          */
int k[3][2] =  {  {1, 1},
                  {2, 2},
                  {3, 3} } ;   /* Эквивалентно {1, 1, 2, 2, 3, 3}          */
```

Начальное значение символьного массива можно задавать в виде строки символов, т. е. текста, заключенного в двойные кавычки.

2.9.3. Указатели

Данные *ссылочного типа* называют *указателем* или *ссылкой*. Значением указателя является *адрес* - номер первой ячейки памяти, занимаемой переменной, константой или функцией. При описании указателя задается тип объектов, адреса которых могут присваиваться этому указателю, т. е. ссылочный тип записывается в виде

тип *

На ПЭВМ указатель занимает 4 байта или 2 байта в зависимости от реализации..

Пусть, например, программа содержит определения переменных:

```
int X, Y, Z[20];     /* целые                          */
float T;
int *P, *Q;          /* указатели данных типа int  */
```

Можно присваивать указателю никуда не показывающий *пустой указатель* NULL (адрес несуществующего объекта), адрес переменной подходящего типа с помощью операции &, значение другого указателя, сравнивать указатели между собой на равенство:

```
Q = NULL;     P = & X;     if (Q==NULL)  Q = P;
```

После приведенных выше операций указатели P и Q равны между собой и оба равны адресу переменной X (*показывают на* X).

Операция *косвенной адресации* * P получает объект, на который показывает указатель P (адрес которого равен P). Чтобы правильно обращаться с объектом *P, транслятору необходимо знать его тип. Поэтому при описании указателя задается тип объектов, на которые он может показывать.

Например, теперь *P, *Q и X обозначают одно и то же, и оператор *P = *Q + 2; равносилен X = X + 2;

После присваиваний

P = &Y; *P = 5;

Y будет равен 5.

Недопустимо присваивание P = &T; , т. к. P имеет тип **int*** и может показывать только на данные типа **int**, а T относится к типу **float.**

Указатели тесно связаны с массивами. *Имя массива* без индексов обозначает указатель начала массива, т. е. Z равно & Z[0] .

Имя массива является *постоянным указателем* и ему нельзя ничего присваивать. Однако, если *имя масссива* является формальным параметром функции, то оно является переменным указателем. Таким образом, в заголовке функции описание параметра

char S[] равносильно **char** *S .

После присваивания

P = Z; (или P = & Z[0];)

с указателем P можно обращаться, как с именем массива, т. е.

P[5] становится равносильным Z[5] .

После присваивания Q=& Z[10]; Q[5] становится равносильным Z[15].

Вообще, если P указывает на Z[j], то

по определению P + k указывает на Z[j+k] (4.4)

*(Z + k) равносильно Z[k] или &Z[k] равносилен Z + k.

Указатели можно складывать с числами, причем эти числа автоматически умножаются на размер объекта, адресуемого указателем.

P[j] равносильно *(P+j) (4.5)

Примеры определения указателей:

на базовый тип **char** *p;
на указатель **char** **t;
одномерный массив **int** x[40];
двумерный массив **float** m[7][30];
массив указателей **char** *t[8];

2.9.4. Строка символов

В ряде языков программирования имеется тип данных string - *строка символов* (иногда говорят просто “строка”), представляющая собой последовательность символов - текст. В языке C в явном виде нет такого типа данных. Имеются лишь *строковая константа*, представляемая в памяти массивом символов с признаком конца в виде нулевого байта '\0': транслятор автоматически добавляет этот байт. Такое представление иногда называют "*ASCIIZ-строка*" - строка символов в коде ASCII с нулем (Zero) в конце.

строка ::= "[символ...]"

Значением строковой константы является адрес (указатель) соответствующего текста (а не сам этот текст).

Пример 2.13

```
char *p;        /* Указатель (адрес) данных типа char          */
. . .
p= "КАИ";      /* Адрес текста из 4 байтов: 'К', 'А', 'И', '\0'   */
printf (p);     /* Эквивалентно  printf ("КАИ");               */
```

Кроме того, стандартная библиотека содержит набор функций над строками символов, рассчитанных на такое же представление строк, как у строковых констант.

Переменные строки в языке С отсутствуют. Их описывают как массивы символов, и программист сам должен заботиться о наличии в них признака конца в виде нулевого байта, если потребуется использовать стандартные функции над строками (см. пример 5.3).

Можно также описать переменную строку символов в виде указателя на символ (типа **char** *), как переменная p в примере 2.13.

Ниже рассмотрены некоторые стандартные функции обработки строк символов (для их использования необходим #**include** <string.h>). Имена всех этих функций начинаются с букв “str” от слова string (подробнее см. приложение 2).

1. **char** * strcpy (**char** *s1, **char** *s2);

Копирование строки s2 в строку s1 (string copy). Необходимо заботиться, чтобы в s1 было достаточно места не только для символов строки s2, но и для признака конца '\0'.

2. **char** * strcmp (**char** *s1, **char** *s2);

Эта функция выполняет сравнение строк: посимвольное сравнение (compare) до первого несовпадения символов. При этом определяется предшествование в *лексикографическом порядке* (как в словарях) в соответствии с кодами символов.

Функция вырабатывает значение меньше 0, если s1 < s2 (s1 в словаре предшествует s2), значение, равное 0, если s1 = s2, и значение больше 0, если s1 > s2 (s1 следует в словаре после s2).

3. **char** * strcat (**char** *s1, **char** *s2);

Прицепление ([con]catenation) строки s2 к концу строки s1 (там должно быть достаточно места). Значение функции равно адресу результата, т. е. строки s1.

4. **int** strlen (**char** *s); Длина строки s (в символах).

2.9.5. Перечислимый тип

Пример 2.14. Данные типа "цвет". Предположим, что требуется иметь дело с переменной, принимающей значения одного из четырех цветов: красный, оранжевый, желтый и зеленый. Простейший способ - обозначить их целыми числами:

```
int cv;        /* 0 -красный, 1 -оранжевый, 2 -желтый, 3 -зеленый */

. . .
cv = 2;        /* желтый */
```

Более наглядна программа с использованием символических констант:

```
#define KRASN   0
#define ORANG   1
#define ZHELT   2
#define ZELEN   3
    ...
cv = ZHELT;       /*  желтый       */
    ...
if (cv < ZELEN)  /* теплый цвет */  ...
```

Перечислимый тип позволяет сделать это короче и нагляднее. Синтаксис этого типа задается следующими правилами (enumerate - перечислять).

перечислимый-тип ::= **enum** [имя-типа] { имя-конст [,имя-конст]... }

имя-конст ::= имя

С использованием перечислимого типа рассматриваемая задача решается следующим образом.

```
enum cvet {KRASN,ORANG,ZHELT,ZELEN};      /* описание типа cvet */
    . . .
enum cvet   cv;      /* определение  переменной  типа   cvet */
    . . .
cv = ZHELT;          /* желтый        */
    ...
if (cv < ZELEN)     /* теплый цвет */  ...
```

Значения (константы) перечислимого типа, обозначаемые именами KRASN, ORANG, ZHELT, ZELEN, хранятся в памяти в виде целых чисел 0, 1, 2 и т. д., соответственно, и занимают объем памяти, как тип **int**.

Пример 2.15. Логический тип из двух значений: “НЕТ и “ДА”.

```
/* Значения:     0   ,   1                                          */
enum logical {NET, DA}  x, y;          /* Описание логического типа
                                          и  переменных x, y     */
enum logical  a, b;              /* Определение логических переменных */
```

```
        . . .
a = NET;
        . . .
if (! x || y && a) . . .
```

Пример 2.16. Тип “День недели”: понедельник, вторник, среда, …, воскресенье .

```
enum den_nedeli {pn, vt, sr, ch, pt, sb, vs}    x, y;
        . . .
if (x < sb) . . .          /* x - рабочий день */
```

2.9.6. Структура

*Структура (*в других языках используют термин *запись)* - это совокупность поименованных элементов (*полей*) разных типов.

```
структурный-тип     ::= struct [имя-типа]  { описание-полей ... }
описание-полей  ::=  тип  описатель-поля[, описатель-поля]... ;
описатель-поля  ::=  описатель
поле-структуры  ::=  поле
поле                ::=  адрес . имя-поля | операнд -> имя-поля
имя-поля            ::=  имя
```

Пример 2.17. Анкета. Требуется представлять в программе анкеты 30 человек, содержащие поля: фамилия, пол, год рождения и вес. Ниже приведен фрагмент программы с определением необходимых данных.

```
struct anketa                    /* тип anketa: информация о человеке  */
{  char fam[20];                 /* фамилия  (20 байтов)               */
   enum {ZHEN, MUZH} pol;        /* пол: женский / мужской             */
   int  godr;                    /* год рождения                       */
   float ves;                    /* вес                                */
}      ank[30];                  /* массив из 30 структур типа anketa  */
          . . .
struct anketa  x, y;             /* x, y - структуры типа anketa       */
struct anketa  *p;               /* указатель (адрес) структуры anketa */
          . . .
ank[2].pol = MUZH;   x.godr = 1975;   y.godr = x.godr + 3;
p = & ank[5];
p->ves = 78.5;                   /* эквивалентно     (*p).ves = 78.5;  */
strcpy (x.fam, "Петров");        /* Копировать "Петров" в  x.fam       */
```

Недопустимо присваивание: x.fam = "Петров"; (поскольку x.fam - константа: постоянный адрес!), но это можно сделать, если описать поле fam не в виде массива, а как указатель на строку:

```
struct anketa                    /* тип anketa: информация о человеке  */
{  char *fam;                    /* адрес фамилии (2 байта)            */
      . . .
}  x;
      . . .
x.fam = "Петров";
```

Поле структуры само может быть структурой. В приведенном ниже примере структурное поле fio состоит из полей: fam, imya и otch.

```
struct anketa
{  struct
   {   char *fam;                  /* фамилия  */
       char *imya;                 /* имя      */
       char *otch;                 /* отчество  */
   }      fio;                    /* Ф. и. о.          */
   enum {ZHEN, MUZH} pol;         /* пол               */
   int  godr;                     /* год рождения      */
   float ves;                     /* вес               */
}      ank[30];                  /* массив из 30 анкет  */
      . . .
ank[2].fio.fam = "Иванов";     /* поле fam структуры fio анкеты ank[2]  */
```

Поля структуры размещаются в памяти друг за другом. Размер структуры равен сумме размеров ее полей. Иногда он может превышать эту сумму, поскольку в ряде ЭВМ некоторые данные должны начинаться с адреса, кратного двум, четырем и т. п., и в этих случаях между полями приходится оставлять неиспользуемые ячейки памяти до ближайшего адреса нужной кратности.

Допускаются битовые поля, занимающие часть ячейки памяти. Каждое поле структуры имеет определенное смещение (расстояние) от начала структуры. Знак "**.**" обозначает операцию нахождения указанного поля заданной структуры (адрес поля определяется как сумма адреса структуры и смещения поля).

2.9.7. Объединение

Объединение представляет собой тип данных, который содержит значения разных типов (из заданного перечня объединяемых типов).

Для объединений используются обозначения, как для структур, только служебное слово **struct** заменяется на **union**. Разница между объединением и структурой состоит в том, что поля объединения не присутствуют в памяти одновременно, как у структуры. Для объединения выделяется область

памяти, равная максимальному из его полей (а не их сумме). Поле определяет возможный способ интерпретации содержимого этой области (как значения соответствующего типа).

объединенный-тип ::= **union** [имя-типа] { описание-полей ... }
поле-объединения ::= поле

Пример 2.18. Использование объединения. Ниже приведено описание данных, занимающих 4 байта, которые будут рассматриваться по-разному в зависимости от используемого имени. Имя cel обозначает первые два байта этих данных, рассматриваемые как значение переменной типа **int** (если тип **int** реализуется двумя байтами). Имя bait обозначает эти же два байта, воспринимаемые как массив из двух символов. В то же время имя vesh вопринимается как переменная типа **float**, занимающая все четыре байта.

```
union int_char_float     /* 4 байта для int, char [2] или float          */
{   int   cel;           /* первые 2 байта, рассматриваемые как int */
    char bait[2];        /* первые 2 байта, как массив из 2 байтов    */
    float vesh;          /* все 4 байта, рассматриваемые как float    */
}  x, y;                 /* x, y, z - переменные типа int_char_float    */
union int_char_float z;
        . . .
x.cel = 5;                /* целое число 5 занимает первые два байта */
            /* x.bait[0], x.bait[1] - младший и старший байты числа 5*/
x.vesh = 4.5;         /* вещественное число 4.5 занимает все 4  байта */
```

Упражнения и задачи

2.1. Составить определение следующих данных.

а) целочисленная величина nom ($0 \le nom \le 50000$);
б) целочисленная величина n ($-20000 \le n \le 30000$);
в) k - количество жителей города;
г) sim - текущая буква текста;
д) s - площадь квартиры с точностью до 0.1 $м^2$;
е) константа pi = π = 3.14159265358 (12 значащих цифр).

2.2. Период времени, начался в h1 часов m1 минут s1 секунд и закончился в h2 часов m2 минут s2 секунд. Составить фрагмент программы вычисления продолжительности этого периода в том же виде: h часов, m минут и s секунд.

а) Период расположен в пределах одних суток;
б) Длительность периода не превышает одних суток,

2.3. Задано трехзначное натуральное число n. Составить программу (фрагмент программы) вычисления следующих величин:

а) значения младшей десятичной цифры числа;

б) значения старшей десятичной цифры числа;

в) числа, полученного из числа n удалением младшей цифры;

г) суммы десятичных цифр числа;

д) произведения десятичных цифр числа;

е) числа, полученного выписыванием в обратном порядке десятичных цифр заданного числа.

2.4. Составить программу (фрагмент программы) вычисления суммы десятичных цифр заданного натурального числа.

2.5. Значением символьной переменной sim является цифра. Присвоить целочисленной переменной z значение этой цифры. Например, если sim='5', то переменной z необходимо присвоить 5.

2.6. Целочисленная переменная z имеет значение от 0 до 9. Символьной переменной sim присвоить цифру (код цифрового символа), числовое значение которой равно z. Например, если z=5, то переменная sim должна получить значение '5'.

2.7. Составить схему данной программы и трассировочную таблицу выполнения этой программы для заданного входного текста.

а) Вход: -2 1 3.1 0.5 2

```
#include <stdio.h>
int x;
void main ()
{  float z, y;
   x=5; y=0;
   while (x > 0)
   {  scanf("%f", &z);
      if (z>1) y=y+z;
      x--;
   }
   printf("%f", y+10);
}
```

б) Вход: -1.5 2 -3 7 0

```
#include <stdio.h>
void main (void)
{  float x, y;
   int r=1;
   scanf("%f", &x);
   if (x!=0)
   {  scanf("%f", &y);
      do
         if (x*y < 0)
         {  x=y; scanf("%f",&y);}
         else { y=0; r=0; }
       while (y != 0);
  }
   if (r)  printf ("Да");
   else   printf ("Нет");
 }
```

в) Вход: 26 марта 2000 г.

```
#include <stdio.h>
char c; int a, b;
void main ()
{  a=0; b=-1;
   do
   {  scanf ("%c", &c);
      if (c==' ' || c=='.') a=0;
```

г) Вход: 45

```
#include <stdio.h>
void main ()
{  int z, v;
   scanf ("%d", &z);
   if (z < 64)
   {  v = 16;
      do
```

```
    else a++;
    if (a>b) b=a;
  } while (c!='.');
  printf (" %d", b);
}
```

```
    {  putchar ('0' + z / v);
       z = z % v;    v = v / 4;
    } while (v > 0);
  }
}
```

2.8. Пусть p, q, r - целочисленные беззнаковые переменные. Составить фрагменты программы для решения следующих задач.

а) Упаковка. Поместить младшие 3 бита переменной p в младшие 3 бита переменной r, младшие 6 битов переменной q - в следующие 6 битов переменной r без изменения остальных битов.

б) Распаковка. Присвоить младшим битам переменной p младшие 3 бита переменной r, а младшим битам переменной q - следующие 6 битов переменной r. Остальные биты p и q обнулить.

2.9. Записать следующие фрагменты программы без помощи оператора **for**.

а)
```
for (j=0;j<10;j++)
  m[j] = 0;
```

б)
```
for (r=-1, n=500; n>0; n--)
{ scanf ("%f",&z);
  if (z<r)  r=z;
 }
```

2.10. Записать следующие фрагменты программы с помощью оператора **for**. Дополнить их определением типов переменных.

а)
```
j=0;
while (j < 80)
{  s[j] = '  ';
   j++;
}
```

б)
```
y=0; m=100;
while (m >= 1)
{  scanf("%f", &x);
   if (x<0)  y = y - x;
   else      y = y+x;
    m = m-1;
 }
```

в)
```
j=0; k=0;
while (t[k]!='.')
{  if (t[k]!='*')
   {   r[j]=t[k];  j++; }
          k++;
}
```

2.11. Составить программу вычисления произведения двух натуральных чисел A и B, не используя операцию умножения.

2.12. Подсчитать объем памяти для данных, определенных следующим образом.

а) **char** a, *b;

б) **unsigned** u[8];

в) **int** m[100][10];

г) **float** x[5][6][2];

д) **int** *p[20];

е)
```
typedef  float   t;
    t    a, b[10], *c[7];
```

ж) **enum** {A, B, C, D} x, y[6];

з) **struct** S

и) **union** U

к) **typedef struct**

```
{  int a;
   char t[30];
   float b;
} x[10], y;
struct S u, *v;
```

```
{  int  x;
   float  y;
   union U *q;
 } a, *b;
```

```
{    int a;
     char t[30];
     float b;
} st;
st  r, ms[10], *q;
```