Языки программирования. Основы С

Алфавит языка C

- Символы для образования ключевых слов и идентификаторов:
  - `a..z, A..Z, _, 0..9`
  - Регистр букв важен!
- Спецсимволы: `, . ! ~ | ^ & #` и др.
- Управляющие последовательности:
  - `\n \t \\ \0` и др.
- /* Комментарии */

Типы данных

- Целые типы:
  - char
  - int
    - квалификаторы short, long
  - квалификаторы signed, unsigned
- Вещественные типы:
  - float
  - double

```
unsigned int a,b;

int b,c;/* подразумевается
signed int b,c; */

unsigned d;/*подразумевается
unsigned int d; */

signed e;/*подразумевается
signed int e; */

i;/*подразумевается
signed int i; */


float f, g;

double h;
```

Размеры типов данных

Тип	Размер (байт)	Диапазон значений
char	1	от -128 до 127
int	2	от -32768 до 32767
short	2	от -32768 до 32767
long	4	от -2 147 483 648 до 2 147 483 647
unsigned shar	1	от 0 до 255
unsigned int	2	от 0 до 65535
unsigned short	2	от 0 до 65535
unsigned long	4	от 0 до 4 294 967 295

Тип	Размер (байт)	Диапазон значений	Примечание
float	4	от 3.14E-38 до 3.14E+38	1 байт для знака, 8 бит для избыточной экспоненты и 23 бита для мантиссы
double	8	от 1.7E-308 до 1.7E+308	1 бит для знака, 11 бит для экспоненты и 53 бита для мантиссы

- Размер переменной (любого типа): `sizeof()`

Перечислимые типы

```
/* flag имеет анонимный тип, значения не указаны явно (0,1,…)*/
enum { FALSE, TRUE } flag;

/* day_of_week имеет анонимный тип, значения указаны явно */
enum {
      MON=1, TUE=2, WED=3, THU=4,
      FRI=5, SAT=6, SUN=7 } day_of_week;

/* errcode – тег типа переменной code, имеющей анонимный тип */
enum errcode {
      ENOMEM=-1, EOVERFLOW=-2, ESRCH=-3, EOTHER=-4 } code;

/* определение переменных с помощью тега типа */
enum errcode code2, code3;
```

```
/* Контроль значений перечислимого типа не производится! */
flag=TRUE; flag=233;
code=FALSE;
day_of_week=code;
```

Массивы

- Индексация на базе `int`, нумерация с 0.

```
int A[100];
b[10];

enum errcode EC[100];

int B[3][3]; /* B[1] - int массив[3] */
int cube[3][3][3]; /* cube[1] - int матрица[3][3] */
float array[5]={0.0,1.5,2.7,3.9,-4.0};
int matrix[3][2]={ {0,1}, {2,3}, {4,5}};

char str[50]="This is a string.";
char str[]="This is a string too.";
```

```
/* Контроль значений индекса массива не производится! */
b[10]=b[11]=22;
matrix[1][2]=7;
EC[-3]=ESRCH;
```

Структуры

```
/* person – тег типа переменных Ivanov, Petrov, имеющих
 анонимный тип */
struct person {
        char * name;
        int age;
        char sex; } Ivanov, Petrov;

Ivanov.name="Иванов И.И.";
Ivanov.age=22;
Ivanov.sex='M';

/* Использование тегов для описания рекурсивных структур */
struct listnode {
        int info;
        struct listnode * next; } leasthead;

struct treenode {
        int info;
        struct treenode * left, * right; } treeroot;
```

Объединения

- Объединения – Pascal-"записи с вариантами", где для всех вариантов выделяется одна и та же область памяти.

```
union {
        char name[30];
        char address[80];
        int age; } info;
union {
        int as_word;
        char as_bytes[2]; } word;

struct {
        int x,y;
        enum { POINT, CIRCLE, RECT } shapetype;
        union {
                int radius;
                int width, height; } shapeinfo;
} shape1, shape2;
```

Поля битов

- Элементом структуры может быть *битовое поле*. Битовое поле рассматривается как целое число, максимальное значение которого определяется длиной поля. Поле нулевой длины обозначает выравнивание на границу следующего слова.
- Структуры битовых полей могут содержать знаковые компоненты. Такие компоненты автоматически размещаются на соответствующих границах слов, при этом некоторые биты слов могут оставаться неиспользованными.
- Вне структур битовые поля объявлять нельзя. Нельзя организовывать массивы битовых полей. Нельзя применять к полям операцию определения адреса.

```
struct {
        unsigned a1 : 1;
        unsigned a2 : 2;
        unsigned a3 : 5;
        unsigned a4 : 2; } example;
```

Интерпретация сложных описаний типов

1. Начать с идентификатора и посмотреть вправо, есть ли `[` или `(`.
2. Если они есть, то интерпретировать эту часть описателя. Затем посмотреть налево, есть ли `*`.
3. Если на любой стадии справа встретится `)`, то применить правила 1. и 2. внутри `()`, а затем продолжить интерпретацию.
4. Интерпретировать спецификатор типа.

Интерпретация сложных описаний типов

```
int * (* apfpi[100])();
/*
 6  5  3 1    2     4
Переменная apfpi (1) - массив из 100 (2) указателей (3)
на функции (4), каждая из которых возвращает указатель (5)
на целое (6).
*/

char * ( * ( * pfpapc)())[10];
/*
7    6   4   2 1      3  5
Переменная pfpapc (1) - указатель (2) на функцию (3),
которая возвращает указатель (4) на массив (5) из 10 элементов,
каждый из которых является указателем (6) на char (7).
*/
```

Синонимы типов

```
typedef enum {FALSE, TRUE} bool_t;
typedef int arr_t[100];
typedef char * string;
typedef char str50[50];
typedef struct { char50 name; … } person_t;

typedef struct person_t persontab_t[100];

typedef struct node {
        person_t info;
        struct node * next; } node_t;

typedef person_t * (* process_fn_t)(person_t *);

typedef struct treenode {
        person_t info;
        process_fn_t process_fn;
        struct treenode *left, *right;
} treenode_t;
```

Константы

```
/* Макроопределение */
#define N 100
#define S(p1,p2,p3) func_with_long_name(p1,p2,p3)
#define begin {
#define end }

/* Переменная с неизменяемым значением */
const float eps=e-5;
const DIM=100;
const char errmsg="Out of memory!";
```

```
/* Константные значения */
#define N 100L /* тип long */
#define n 100l /* тип long */
int A=0x12ab, B=0XFFF; /* 16-ричное */
int C=01234567; /* 8-ричное */
#define BYTE_LIMIT 255U /* unsigned */
```

Выражения

- Унарные (справа налево):
  `unary_sign operand`
- Бинарные (слева направо):
  `operand1 binary_sign operand2`
- Тернарные:
  `operand1 ?  operand2 : operand3`

```
++a;
--a;
*a
&a

a=b=c;
a+=b;
a%=b;

#define MAX(a,b) ((a)>(b) ? (a) : (b))
```

Бинарные операции

Группа операций	Знак	Название операции
Мультипликативные	*	Умножение
	/	Деление
	%	Остаток от деления
Аддитивные	+	Сложение
	-	Вычитание
Операции сдвига	<<	Сдвиг влево
	>>	Сдвиг вправо
Операции отношения	<	Меньше
	<=	Меньше или равно
	>=	Больше или равно
	==	Равно
	!=	Не равно

Бинарные операции

Группа операций	Знак	Название операции
Поразрядные операции	&	Поразрядное И
	|	Поразрядное ИЛИ
	^	Поразрядное исключающее ИЛИ
Логические операции	&&	Логическое И
	||	Логическое ИЛИ
Последовательного вычисления	,	Последовательное вычисление

Бинарные операции

Группа операций	Знак	Название операции
Операции присваивания	=	Присваивание
	*=	Умножение с присваиванием
	/=	Деление с присваиванием
	%=	Остаток от деления с присваиванием
	-=	Вычитание с присваиванием
	+=	Сложение с присваиванием
	<<=	Сдвиг влево с присваиванием
	>>=	Сдвиг вправо присваиванием
	&=	Поразрядное И с присваиванием
	|=	Поразрядное ИЛИ с присваиванием
	^=	Поразрядное исключающее ИЛИ с присваиванием

Преобразование типов

- Явное: `(typename) variable`
- Неявное:
  1. Операнды типа `float` преобразуются к типу double.
  2. Если один операнд `long double`, то второй преобразуется к этому же типу.
  3. Если один операнд `double`, то второй также преобразуется к типу double.
  4. Любые операнды типа `char` и `short` преобразуются к типу `int`.
  5. Любые операнды `unsigned char` или `unsigned short` преобразуются к типу `unsigned int`.
  6. Если один операнд типа `unsigned long`, то второй преобразуется к типу `unsigned long`.
  7. Если один операнд типа `long`, то второй преобразуется к типу `long`.
  8. Если один операнд типа `unsigned int`, то второй операнд преобразуется к этому же типу.

Приоритеты операций

Приоритет	Знак операции	Типы операции	Порядок выполнения
1	- ~ ! * & ++ -- sizeof приведение типов	Унарные	Справа налево
2	() [] . ->	Выражение	Слева направо
3	* / %	Мультипликативные	Слева направо
4	+ -	Аддитивные	
5	<< >>	Сдвиг	
6	< > <= >=	Отношение	
7	== !=	Отношение (равенство)	
8	&	Поразрядное И	

Приоритеты операций

Приоритет	Знак операции	Типы операции	Порядок выполнения
9	^	Поразрядное исключающее ИЛИ	Слева направо
10	|	Поразрядное ИЛИ	
11	&&	Логическое И	
12	||	Логическое ИЛИ	
13	? :	Условная	
14	= *= /= %= += -= &= |= >>= <<= ^=	Простое и составное присваивание	Справа налево
15	,	Последовательное вычисление	Слева направо

Операторы

- Пустой: ;
- Составной : {лок. переменные; список операторов}
- Оператор-выражение
- Условные
  - `if(выражение) оператор1 [else оператор2];`
  - `switch(выражение)`
    ```
    {case константа1: список1; break;
     case константа2: список2; break;
     …
     default: списокN;}
    ```
- Циклические
  - `for(выражение1;выражение2;выражение3) оператор;`
  - `while(выражение) оператор;`
  - `do оператор while(выражение);`
- Операторы перехода
  - `break` – завершение цикла или `switch`;
  - `continue` – переход к следующей итерации цикла;
  - `goto метка` – переход к оператору, помеченному меткой;
  - `return` – завершение текущей функции.

Команды препроцессора

- #define имя последовательностьЛексем

  #define TRUE = 1 // неверно, т.к. TRUE равна “= 1”

  #define имя (V1,V2,…VN) последовательностьЛексем
- #include <имя файла> // или “имя файла”
- #ifdef имяПеременной

  операторы

  #endif
- #if
- #undef
- #else

Подпрограммы

- тип имяФункции(параметры){тело}
- void имяФункции(параметры){тело} // процедура

Передача параметров

- по значению:

```
int Cube(int n)
{ return n*n*n;}
```

- по ссылке:

```
void Cube2(int *nPtr)
{ *nPtr = *nPtr * *nPtr * *nPtr}
// вызов Cube2(&number)
```

Функции распределения памяти

- #include <malloc.h>
- void *malloc(числоБайтов)
  // выделение динам. памяти
- ptrvar = (newtype*) malloc(sizeof(newtype));
- int free(char mallocptr)