# *Структура программы на языке Си*

```
#include <системные .h файлы>  // директива препроцессора вставляет
    текст
                                  //  внешнего файла
#include “пользовательские .h файлы”   //  директива препроцессора
#define  аргумент1  аргумент2   //  директива выполнения макроописаний
#include <locale.h>                  // требуется для   setlocale
```

**Необязательный раздел описания вспомогательных функций (прототипы функций)**

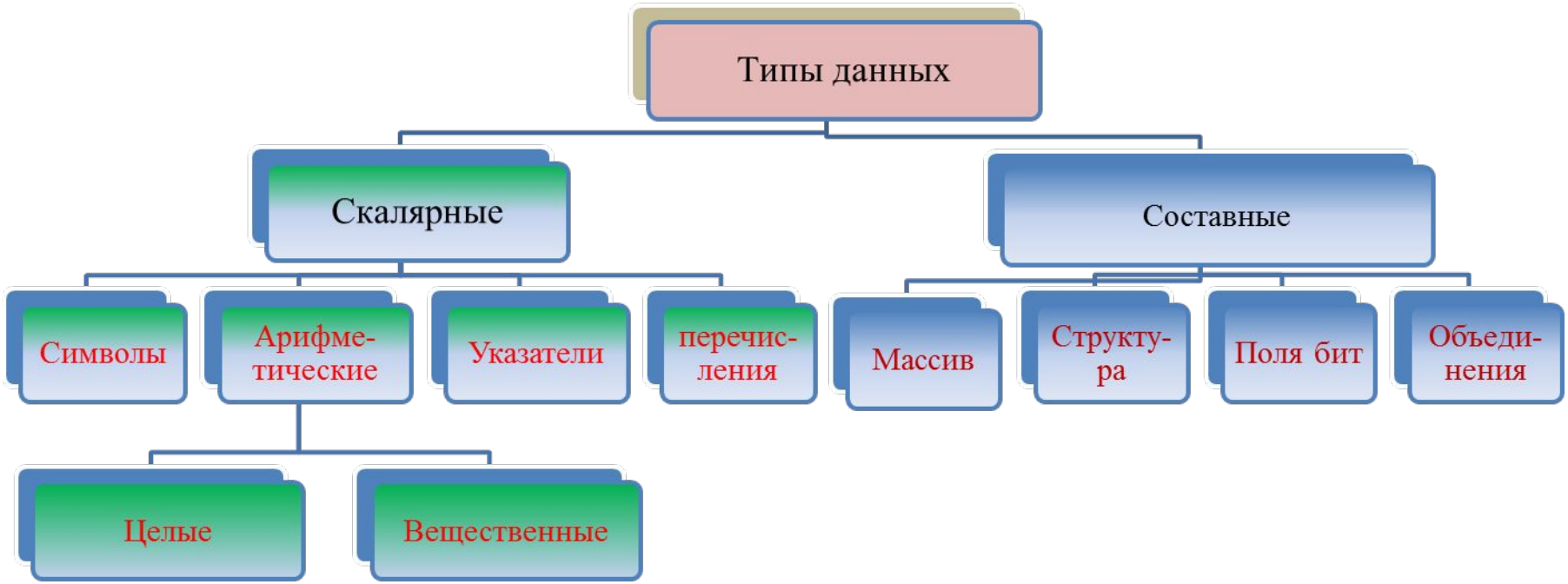
**Раздел описания глобальных данных**

```
int  main()                                   //  заголовок основной функции
{   setlocale(LC_ALL,"Russian");  // русификация вывода информации на экран
```

**Раздел описания локальных данных**

```
      операторы языка Си
      вызовы вспомогательных функций (если они есть)
    return   0;
}
```

# *Типы данных языка Си*



## *Данные целого типа*

| название типа | число байт | диапазон значений |
|---|---|---|
| **char** | 1 | -128...127 |
| **signed char** | 1 | -128...127 |
| **unsigned char** | 1 | 0...255 |
| **int** | 2,4 | -32 768...32 767 |
| **signed int** | 2,4 | -32 768...32 767 |
| **unsigned int** | 2,4 | 0...65 535 |
| **signed short int** | 2 | -32 768...32 767 |
| **unsigned short int** | 2 | 0...65 535 |
| **long int** | 4 | -147483648 … 2147483647 |
| **signed long int** | 4 | - “ - |
| **unsigned long int** | 4 | - “ - |

## *Данные вещественного типа*

| название типа | число байт | диапазон значений |
|---|---|---|
| **float** | 4 | 1.17E-38 … 3.37E+38 |
| **double** | 8 | 2.23E-308 … 1.67E+308 |
| **long double** | 10 | 3.37E-4932 … 1.2E+4932 |

# Перечисляемый тип (enum)

Ключевое слово enum используется для объявления перечисляемого типа. Он предназначен для описания констант (типа int) из некоторого заданного множества, например:

```
enum number {one,two,four,nine} ;
enum number m;
```

Переменная m может принимать любое значение из списка констант перечисленных в фигурных скобках. Каждому значению из списка соответствует целое десятичное число, начиная с нуля. Каждая следующая имеет значение на единицу больше, чем предыдущая:

one=1, two=2, four=3 и так далее.

Определение переменных можно выполнить и при объявлении типа, например:

```
enum number {one,two,four,nine} i1=one, i2=four;
```

Перечисление может быть описано и без задания имени типа. Имена в различных перечислениях должны отличаться друг от друга. Значения внутри одного перечисления могут совпадать:

```
enum number {one,two=one,four=4,six=4,nine} i1=one, i2=two;
```

В этом случае переменные i1 и i2 будут равны обе нулю и ассоциироваться с константой one. Константы four и six будут равны четырем .

В перечислении константам можно задавать значения не по порядку, при этом если не все значения констант явно специфицированы, то они продолжают прогрессию, начиная от последнего специфициро-ванного значения:

```
enum number{one= 2,two,four= two+one-1,six= two+3} i1=two, i2= four;
```

В этом случае значения именованных констант будут следующими:

```
one= 2, two=3, four= 4, six= 6.
```

# *Операции языка Си*

## Арифметические операции

| | | | |
|---|---|---|---|
| **+** | сложение | **a= b + c;** | если b=6, c=4, то a=10 |
| **-** | вычитание | **a= b - c;** | если b=3, c=8, то a= -5 |
| **-** | изменение знака | **a= - b;** | если b=132, то a= -132 |
| ***** | умножение | **a= b * c;** | если b=2, c=4, то a= 8 |
| **/** | деление | **a= b / c;** | если b=7.0, c=2.0, то a= 3.5 |
| **%** | остат. от деления | **a= b % c;** | если b=10, c=3, то a= 1 |

**+=, -=, *=, /=, %=** - составная операция присваивания

**++** Унарный инкремент a++ (++a) значение a увеличивается на единицу

**--** Унарный декремент a-- (--a) значение a уменьшается на единицу

# Логические операции

**&&** Логическое “И” (конъюнкция) a= b && c;

если b и c не равны нулю, то a=1, иначе a=0

|| Логическое “ИЛИ” (дизъюнкция) a= b || c;

если b и c равны нулю, то a=0, иначе a=1

**!** Логическое отрицание (инверсия) a= **!** b;

если b равно 0, то a=1, если b не равно 0, то a=0

Пример:

```
if( a>b && c<3 || a+3>c)  ……
if(! a)  …….
```

# Поразрядные операции

**&** Поразрядная конъюнкция a= b & c;

если оба сравниваемых бита единицы, то бит результата равен 1, в противном случае – 0.

Если b=1010, c=0110, то a=0010

| Поразрядная дизъюнкция a= b | c;

если любой (или оба) из сравниваемых битов равен 1, то бит результа-та устанавливается в 1, в противном случае - в нуль.

Если b=1010, c=0110, то a=1110

^ Поразрядное “исключающее ИЛИ” a= b ^ c;

если один из сравниваемых битов равен 0, а второй бит равен 1, то бит результата равен 1, в противном случае 0.

Если b=1010, c=0110, то a=1100

~ Поразрядная инверсия a= ~ b;

если b=1010, то a=0101

# Сдвиговые перации

**<<** Сдвиг влево a= b << c;

осуществляется сдвиг значения **b** влево на **c** разрядов; в освобо-дившиеся разряды заносятся нули.

Если b=1011, c=2, то a=1100

**>>** Сдвиг вправо a= b >> c;

осуществляется сдвиг значения **b** вправо на **c** разрядов; в освобо-дившиеся разряды заносятся нули, если **b** имеет тип **unsigned** и копии знакового бита в противном случае.

Если b=1011, c=2, то a=0010

# Операции сравнения

== Сравнение на равенство a == b;

вырабатывается 1, если **a** равно **b**, и 0 – в противном случае

> , >= Больше , (Больше или равно) a > b; a>=b;

вырабатывается 1, если **a** больше (больше или равно) **b**, и 0 – в противном случае

< , <= Меньше , (Меньше или равно) a < b; a<=b;

вырабатывается 1, если **a** меньше (меньше или равно) **b**, и 0 – в противном случае

!= Не равно a != b;

вырабатывается 1, если **a** не равно **b**, и 0 – в противном случае

## Операции адресации и разадресации

**&** Адресация ptr= & b;

**ptr** присваивается адрес переменной **b** ;

***** Разадресация a= * ptr;

Переменной **a** присваивается значение содержащееся по адресу **ptr** ;

## Операция последовательного вычисления

**,** Запятая a = (c-- , ++b);

вычисляет операнда слева направо. Результат операции имеет значение и тип второго операнда. Если c=2, b=3, то a=4, c=1, b=4

## Операция условного выражения

**?:** Условная операция a= (b<0) ? (-b) : (b);

**a** присваивается абсолютное значение **b**

## Size-операция

**Sizeof** Определение размера в байтах a= sizeof(int);

# Приоритет операций

В таблице показан приоритет (порядок вычисления) операций языка С(С++). Операции упорядочены по приоритету сверху вниз.

| Операция | Приоритет |
|---|---|
| () [] -> . | слева направо |
| ! ~ ++ — + - * & sizeof | справа налево |
| * / % | слева направо |
| + - | слева направо |
| << >> | слева направо |
| < <= > >= | слева направо |
| == != | слева направо |
| & | слева направо |
| ^ | слева направо |
| \| | слева направо |
| && | слева направо |
| \|\| | слева направо |
| ?: | справа налево |
| = += -= *= /= %= &= ^= \|= <<= >>= | слева направо |

# *Ввод и вывод информации*

## Вывод информации

В языке С вывод информации обеспечивается функциями:
**printf(), putchar(), puts().**

**int printf("управляющая строка", аргумент1, аргумент2, ... );**

**Управляющая строка** представляет собой символьную строку, заключенную в кавычки ("xxxx") и определяющую порядок и формат печати выводимой информации. параметра в поле вывода.

**Управляющая строка** может содержать информацию вида:

1) обычная текстовая информация;

2) спецификация преобразования, согласно которой осуществляется вывод одного из аргументов, содержащихся в списке аргументов. Формат спецификации:

**% [знак] [размер поля вывода ][точность]**

**символ_преобразования**

**Знак,** расположенный после %, указывает на необходимость выравни-вания выводимого параметра в поле вывода и имеет следующий смысл**:**

| | |
|---|---|
| - | Выравнивание влево выводимого числа в пределах выделенного поля, по умолчанию производится выравнивание вправо. |
| + | Выводится знак + перед положительным числом |
| пробел | Выводится знак пробела перед положительным числом |
| 0 | Заполняет поле нулями |

## Символы преобразования:

| символ | Представление данных |
|---|---|
| **d, i** | Целое десятичное число |
| **u** | Целое беззнаковое десятичное число |
| **o** | Целое беззнаковое восьмеричное число |
| **x, X** | Целое беззнаковое шеснадцатеричное число |
| **f** | Число с плавающей запятой в записи с фиксированной десятичной точкой |
| **e, E** | Значение со знаком в форме [-]d.ddde[+\|-]ddd |
| **c** | Одиночный символ |
| **s** | Символьная строка |
| **p** | Указатель |

***Перед типом формата могут стоять следующие*** **модификаторы:**

| | |
|---|---|
| **h** | Если тип формата – **d, i, o, x** или **X**, то тип параметра - **short int**.<br>При типе формата **u** тип параметра – **unsigned short int** |
| **l** | При типе формата – **d, i, o, x** или **X** тип параметра – **long int**, при **u - unsigned long**.<br>При типе формата – **e, E, f, g** или **G** тип параметра – **double.** |

# Ввод информации

**Ввод информации обеспечивается функциями**
**scanf, getch, getchar , gets и др.**

**int scanf("управляющая строка", адрес аргумента1, адрес аргумента2, ... );**

В функции scanf в качестве аргументов используются **только указатели** на переменные.

В отличие от функции printf, в **спецификации** преобразования функции scanf:

- отсутствует спецификация %g;
- спецификации %f и %e эквивалентны;
- для ввода целых чисел типа short применяется спецификация %p.

# *Операторы языка Си*

- условные операторы: условия if и оператор выбора switch;
- операторы цикла: for , while , do while;
- операторы перехода: break , continue , return , goto;
- другие операторы: оператор "выражение", пустой оператор.

***Составной оператор*** – это несколько операторов объединенных в блок с помощью фигурных скобок { }, или разделенных запятой. Такой блок можно рассматривать как один оператор.

***Пустой оператор*** – состоит только из знака ; .

# *Операторы цикла*

***В языке Си имеется три оператора цикла:***

- ***for***
- ***while***
- ***do while***

## *Оператор for (цикл с фиксированным числом повторов)*

**for(выражение1; выражение2; выражение3) тело цикла;**

Тело цикла составляют одна либо некоторое подмножество инструк-ций, заключенных в фигурные скобки.

**выражение 1** служит для инициализации переменной цикла.

**выражение 2** устанавливает условие, при котором цикл повторяется.

**выражение 3** выполняет изменение переменной цикла.

**Пример:**

```
a=0;
for(i=0; i<5;i++)
a=a+i;      // в цикле одна инструкция
```

```
a=0;  b=1;
for(i=0; i<5;i++)
{  a=a+i;     // в цикле несколько
   b=b*i;       // инструкций
}
```

## ***Оператор while (цикл с предварительной проверкой условия)***

**while(логическое выражение) тело цикла;**

Тело цикла составляют одна либо некоторое подмножество инструкций, заключенных в фигурные скобки. **Логическое выражение** проверяется в начале каждой очередной итерации цикла.

**Пример:**

```
a=0;   i=0;
while(i<5)
a=a+i++;   // в цикле одна инструкция
```

несколько

```
a=0;  b=1;
while(i<5)
{  a=a+i;   // в цикле

   b=b*i;  // инструкций
   i++;
```

## ***Оператор do …while (цикл с проверкой условия в конце итерации цикла)***

**do . . . while(логическое выражение) тело цикла;**

Тело цикла составляют одна либо некоторое подмножество инструкций, заключенных в фигурные скобки. **Логическое выраже-ние** проверяется после окончания каждой очередной итерации цикла.

**Пример:**

```
a=0;   i=0;
do
  a=a+i++;   // в цикле одна инструкция
while(i<5);
```

```
a=0;  b=1;  i=0;
do
{  a=a+i;        // в теле цикла
   b=b*i;        // несколько
   i++;          // инструкций
}
while(i<5);
```