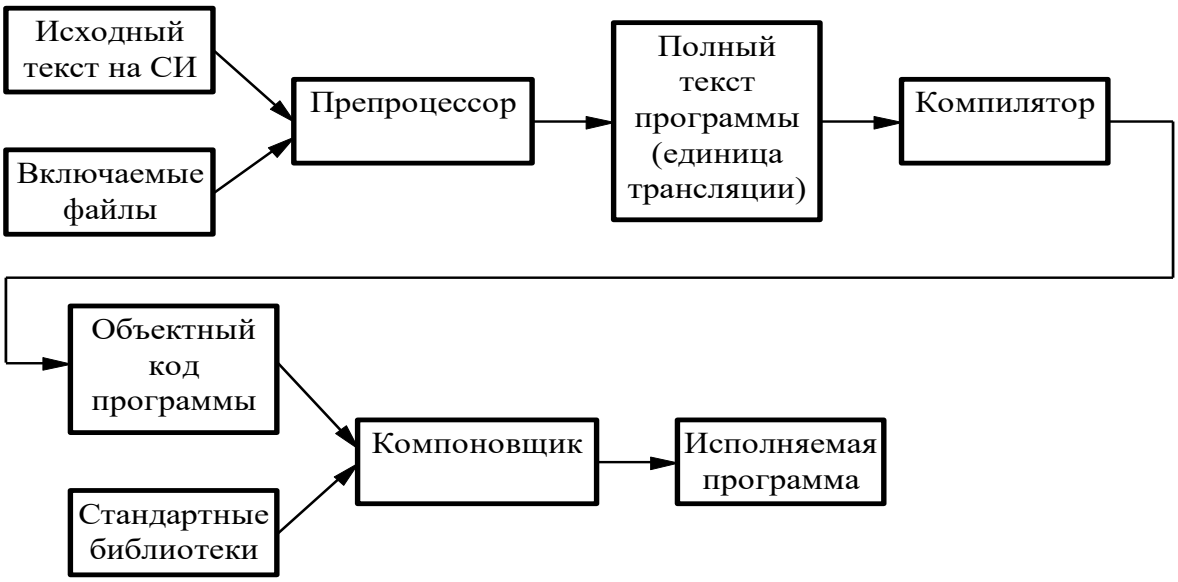
Структура и компоненты простой программы



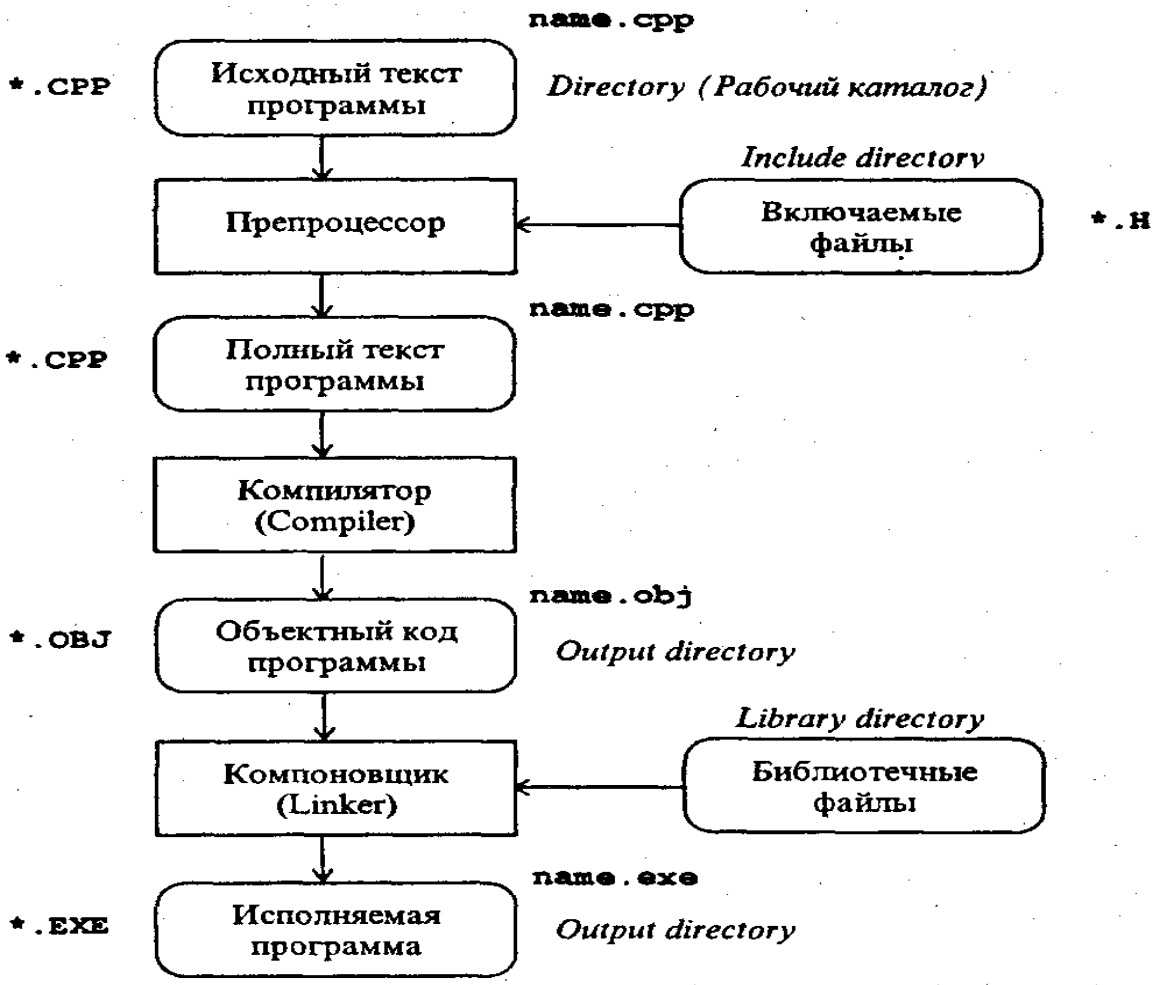
Функция с фиксированным именем **main** является главной функцией программы, без которой программа не может быть выполнена. Имя этой главной функции для всех программ одинаково и не может выбираться произвольно. Исходный текст программы в случае, когда программа состоит только из одной функции, имеет вид:

```
директивы препроцессора
int main( )
{ определения_ объектов;
исполняемые _операторы;
}
```

Определения вводят объекты, необходимые для представления в программе обрабатываемых данных. Примером таких объектов служат именованные константы и переменные разных типов.

Описания уведомляют компилятор о свойствах и именах объектов и функций, определенных в других частях программы. *Операторы* определяют действия программы на каждом шаге ее выполнения.

Для подключения к программе описаний средств **ввода-вывода** из стандартной библиотеки компилятора используется директива **#include <stdio.h>.**



Кое-что о СИ

Идентификатор. Последовательность букв, цифр и символов подчеркивания "_" , начинающаяся с буквы или символа подчеркивания:

```
KCM_16,
size88,
_MIN,
TIME,
time.
```

Прописные и строчные буквы различаются, т.е. два последних идентификатора различны.

Стандартные данные

Целые числа могут быть представлены в 10-тичной, 8-ричной и 16-ричной системах счисления.

Десятичная система используется для чисел со знаком и без, 8-чная и 16-чная системы – для чисел без знака.

Число в 8-ричной системе счисления начинается всегда с 0 (ноль), за которым следует некоторое количество 8-ричных цифр 0..7, например, 0731.

Число в 16-ричной системе счисления начинается с 0x за которым следует некоторое количество 16-ричных цифр 0..9, a..f (A..F), например,0x9AF7.

Для представления символов используется 1 байт. Поэтому с помощью восьми бит можно закодировать 2^8=256 различных символов. Каждому символу соответствует свой код, равный порядковому номеру символа в таблице символов. Коды изменяются от 0 до 255. В компьютере символы представляются их кодами, поэтому они описываются целым типом.

Существуют 3 способа представления символов в программе на С:

1) в апострофах, например, ‘A’;
2) ‘\код символа’ или “\код символа” Например “\49” = ‘1’ = “\061” = ‘\0x31’
3) С применением специальных обозначений для некоторых символов:

‘\b’ – забой, BackSpace, возврат на шаг,

‘\n’ – новая строка, перевод строки , Enter;

‘\t’– горизонтальная табуляция (4-6 пробелов)(9, 11 коды);

‘\0’ – нулевой байт, нулевой символ;

Строковая константа – набор символов, заключенных в кавычки “, например, “ABC”.

Для представления строковой константы отводится количество байт, равное числу символов в ней + 1 байт под нулевой байт ‘\0’, который заполняется восемью нулями, например:

“12345” - набор символов. ‘1’, ‘2’, ‘3’, ‘4’, ‘5’‘\0’, т.е. всего 6 байт 48 бит.

Основные типы данных

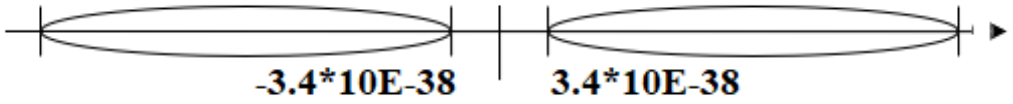
`void`– пустой, не имеющий значения.

Перед указанием типа можно использовать следующие модификаторы: signed – знаковый, unsigned – не знаковый. Эти два модификатора применяются только для **char** и **int**. Signed является необязательным модификатором, т.к. целые числа по умолчанию считаются знаковыми. **int** = **signed int** = **signed**.

Еще существует модификатор **long** (длинный), который можно применить к **int** и **double**.

Тип данных	Размер, бит	Диапазон значений
unsigned char	8	0 ... 255
char	8	-128 ...127
unsigned int	16	0 ... 65535
int	16	-32768 ... 32767
unsigned long	32	0 ... 4294967295
long	32	-2147483648 ... 2147483647
float	32	3.4E-38 ... 3.4E+38
double	64	1.7E-308 ... 1.7E+308
long double	80	3.4E-4932 ... 1.1E+4932

Приближенный «0» в типе **float** равен $\pm 3.4*10^{-38}$.



Приближенный 0 в **long double** $\pm 3.4*10^{-4932}$.

Переменные

Переменная – именованная область памяти, в которую можно записать значение и из которой можно прочитать значение.

Объявление переменных

```
char symbol, cc;
unsigned char code;
int number, row;
float x,y,z;
long intgr;
```

Область доступности переменной определяется блоком, в котором она описана. Блок ограничивается {}.

Инициализация переменных. В соответствии с синтаксисом языка переменные автоматической памяти после определения по умолчанию имеют неопределенные значения. Надеяться на то, что они равны, например, 0, нельзя. В отличие от присваивания, которое осуществляется в процессе выполнения программы, инициализация выполняется при выделении для переменной участка памяти. Примеры определений с инициализацией:

```
float pi=3.1415, cc=1.23;
```

Знаки операций. Для формирования и последующего вычисления выражений используются операции. Для изображения одной операций в большинстве случаев используется несколько символов.

За исключением операций "[ ]", "( )" и "?:", все знаки операций распознаются компилятором как отдельные лексемы. В зависимости от контекста одна и та же лексема может обозначать разные операции, т.е. один и тот же знак операции может употребляться в различных выражениях и по-

разному интерпретироваться в зависимости от контекста. Например, бинарная операция & - это поразрядная конъюнкция, а унарная операция & -это операция получения адреса.

Ранг	Операции	Ассоциа-тивность
1	( ) [ ] -> .	->
2	! ~ + - ++ -- & * *(mun)* sizeof	<-
3	* / % (мультипликативные бинарные)	->
4	+ - (аддитивные бинарные)	->
5	<< >> (поразрядного сдвига для целых)	->
6	< <= >= > (отношения)	->
7	== != (отношения)	->
8	& (поразрядная конъюнкция "И")	->
9	^ (поразрядное исключающее "ИЛИ")	->
10	| (поразрядная дизъюнкция "ИЛИ")	->
11	&& (конъюнкция "И")	->
12	|| (дизъюнкция "ИЛИ")	->
13	? : (условная операция)	<-
14	= *= /= %= += -= &= ^= |= <<= >>=	<-
15	, (операция "запятая")	->

Мультипликативные операции:

* - умножение операндов арифметического типа (ранг 3);

/ - деление операндов арифметического типа.

При целочисленных операндах абсолютное значение результата округляется до целого. Например, 20/3 равно 6,

% - получение остатка от деления целочисленных операндов (деление по модулю - ранг 3).

Логические бинарные операции: **&&** **||**

Унарная операция отрицания '**!**'.

Результаты отношений и логических операций:

```
3<5  равняется 1;
3>5  равняется 0;
3==5      равняется 0;
3!=5      равняется 1;
```

```
3!=5 || 3==3   равняется 1;
3+4>5 && 3+5>4 && 4+5>3  равняется 1.
```

Некоторые приемы алгоритмизации

Обмен значениями между двумя переменными A и B :

```
C = A          A = B          B = C
```

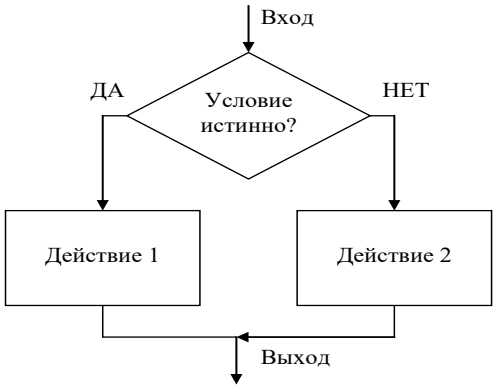
Оператор выражение `a=cos(b * 5);`

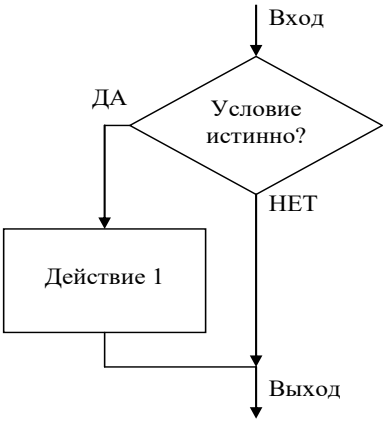
Пустой оператор ;

Составной оператор { оператор1; оператор2; …; операторN; }

Условный оператор if

```
if (выражение_условие)оператор_1; else оператор_2;
if (выражение_условие) оператор;
```

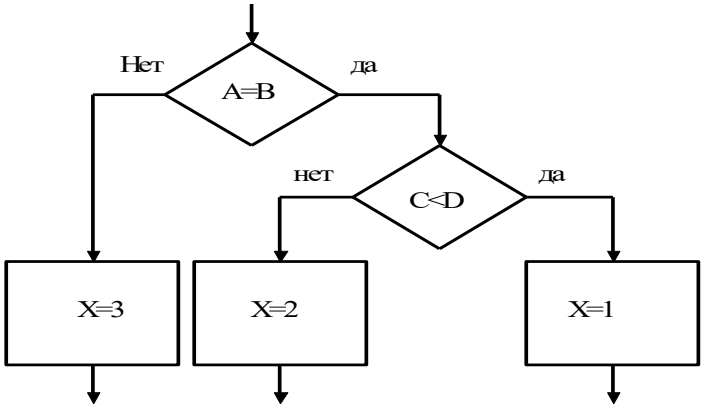




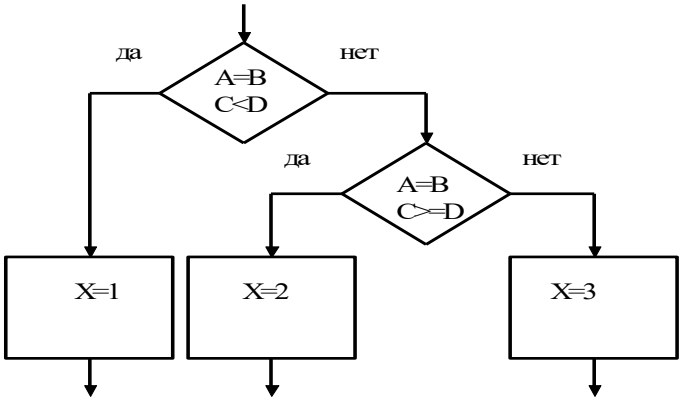

```
if (выражение_условие){ оператор_11;оператор_12; }
     else { оператор_21;оператор_22; }
```

Пример1:

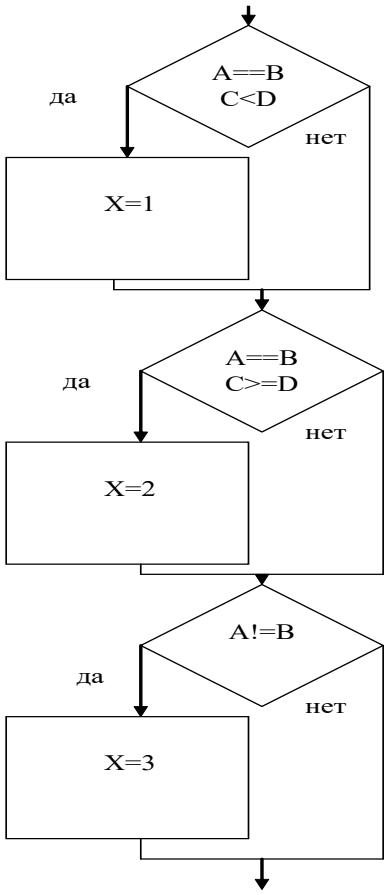
```
if (A==B)
     if (C<D)  X=1;
          else X=2;
     else X=3;
```



```
if (A==B && C<D) X=1;
     else if (A==B && C>=D) X=2;
               else  X=3;
```




```
if (A==B && C<D) X=1;
if (A==B && C>=D) X=2;
if (A!=B)  X=3;
```



Операторы цикла

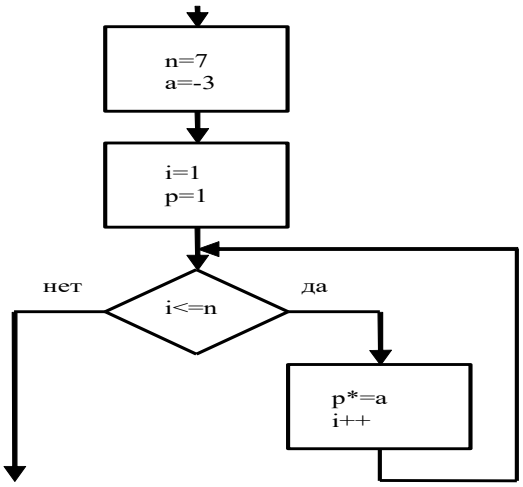
1) while (условие)

оператор; // тело_цикла

В качестве *выражения_условия* чаще всего используется отношение или логическое выражение. Если оно истинно, т.е. не равно 0, то тело цикла выполняется до тех пор, пока *выражение_условие* не станет ложным. Проверка истинности выражения осуществляется до каждого выполнения тела цикла (до каждой итерации). Таким образом, для заведомо ложного *выражения_условия* тело цикла не выполнится ни разу. *Выражение_условие* может быть и арифметическим выражением. В этом случае цикл выполняется, пока значение *выражения_условия* не равно 0.

Пример: Вычислить A^N*:*

```
...
n=7;a=-3;
i=1; p=1;
while (i<=n)
      {p*=a;
      i++;
      }
...
```



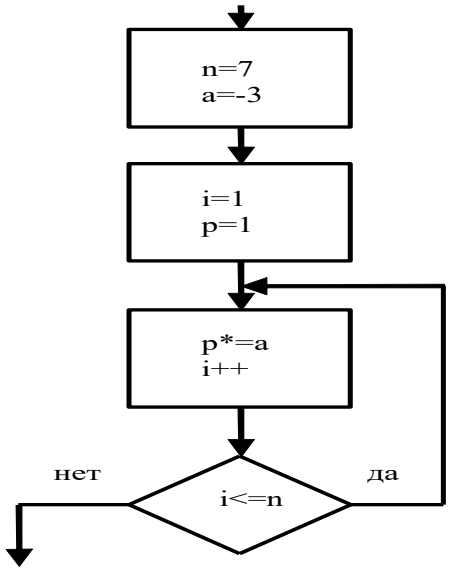
2) do оператор; **//** цикл с постусловием

while (условие);

Выражение_условие логическое или арифметическое, как и в цикле **while.** В цикле **do** тело цикла всегда выполняется по крайней мере один раз. После каждого выполнения тела цикла проверяется истинность *выражения_условия* (на равенство 0), и если оно ложно (т.е. равно 0), то цикл заканчивается. В противном случае тело цикла выполняется вновь.

Пример: Вычислить A^N:

```
...n=7;a=-3;
i=1; p=1;
do { p*=a;
      i++;
      }
while (i<=n);...
```



3) for (выражение1; выражение2 условие**; выражение3) оператор;**

Выражение_1 определяет действия, выполняемые до начала цикла, т.е. задает начальные условия для цикла; чаще всего это выражение присваивания.

Выражение2 условие - обычно логическое или арифметическое. Оно определяет условия окончания или продолжения цикла. Если оно истинно (т.е. не равно 0), то выполняется тело цикла, а затем вычисляется *выражение_3.*

Выражение_3 обычно задает необходимые для следующей итерации изменения параметров или любых переменных тела цикла.

После выполнения *выражения_3* вычисляется истинность *выражения_условия,* и все повторяется...

Таким образом, *выражение_1* вычисляется только один раз, а *выражение_условие* и *выражение_3* вычисляются после каждого выполнения тела цикла.

Цикл продолжается до тех пор, пока не станет ложным *выражение_условие.*

Любое из трех, любые два или все три выражения в операторе **for** могут отсутствовать, но разделяющие их символы ";" должны присутствовать всегда.

Если отсутствует *выражение_условие,* то считается, что оно истинно и нужны специальные средства для выхода из цикла.

Проиллюстрируем особенности трех типов цикла на примере вычисления приближенного значения $e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots = \sum_{n=0}^{\infty} \frac{x^n}{n!}$ для заданного значения x.

Вычисления будем продолжать до тех пор, пока очередной член ряда остается больше заданной точности.

Обозначим точность через `eps`, результат - `b`, очередной член ряда - `r`, номер члена ряда - `i`.

Для получения i-го члена ряда нужно (i-1)-й член умножить на `x` и разделить на `i`, что позволяет исключить операцию возведения в степень и явное вычисление факториала.

Опустив определения переменных, операторы ввода и проверки исходных данных, а также вывода результатов, запишем три фрагмента программ.

```
/* Цикл с предусловием */
i = 2;
b = 1.0;
r = x;
while( r >= eps || r <= -eps )
     {b = b + r;
     r = r * x/ i;
     i++; }
```

Так как проверка точности проводится до выполнения тела цикла, то для окончания цикла абсолютное значение очередного члена должно быть меньше или равно заданной точности.

```
/* Цикл с постусловием */
i = 1;
b = 0.0;
r = 1.0;
do {b = b + r;
     r = r * x/ i;
     i++;       }
while( r > eps || r < -eps );
```

Так как проверка точности осуществляется после выполнения тела цикла, то условие окончания цикла - абсолютное значение очередного члена строго меньше заданной точности. Соответствующие изменения внесены и в операторы, выполняемые до цикла.

```
/* Параметрический цикл */
i = 2;
b = 1.0 ;
r = x;
for(  ; r > eps || r < -eps ; )
     {b = b + r;
     r = r * x/ i;
     i = i + 1 ;    }
```

Условие окончания параметрического цикла такое же, как и в цикле **while.**

Все три цикла записаны по возможности одинаково, чтобы подчеркнуть сходство циклов.

Однако в данном примере цикл **for** имеет существенные преимущества.

В заголовок цикла (в качестве *выражения_1)* можно ввести инициализацию всех переменных:

```
for ( i = 2, b = 1.0, r = x; r > eps || r < -eps;  )
{b = b + r;
r = r * x/ i;
i = i + 1;     }
```

В *выражение_3* можно включать операцию изменения счетчика членов ряда:

```
for( i = 2, b  = 1.0, r = x; r > eps || r < -eps;  i++)
{b = b + r;
r = r * x/ i;  }
```

Можно еще более усложнить заголовок, перенеся в него все операторы тела цикла:

```
for ( i=2, b=1.0, r=x;   r>eps || r<-eps;    b += r, r *= x/ i, i++ );
```

В данном случае тело цикла - пустой оператор.

Для сокращения *выражения_3* в нем использованы составные операции присваивания и операция ++.