Структура программы на языке C:

1) Директивы препроцессора;

2) Объявление функций, используемых и определенных в программе;

3)Управляющая функция main(), в которой находится основной код программы, должна возвращать 0, как знак, что программа выполнилась без ошибок;

4)Определение функций, используемых в программе;

Пример:

1. #include <stdio.h>
2. #define NUM 10
3. int sum(int a, int b);
4. int main()
5. {
6. int n = 7;
7. int res = sum(NUM, n);
8. printf("res = %d", res);
9. return 0;
10. }
11. int sum(int a, int b){
12. return a+b;
13. }

Этапы компилляции:

1) Препроцессинг- исходные файлы расширяются кодом библиотек, вместо макросов препроцессора подставляются их значения, обработка других директив препроцессора.

2) Лексический анализ - код преобразуется в лексемы;

3) Синтаксический анализ: код проверяется на наличие синтаксических ошибок, создается дерево разбора.

4) Семантический анализ: Дерево разбора обрабатывается с целью установления семантики кода(его смысла);

5) Оптимизация: дерево разбора оптимизируется с сохранением семантики программы(смысла), удаляются ненужные конструкции и т.д.

6)Создаются объектные файлы исходных. На этом этапе отсутсвуют связи между файлами и подключенные библиотеки, а также код запуска;

7) Линковка(сборка): создается итоговый файл, со всеми связями, библиотеками и кодом запуска;

Директивы препроцессора:

#define MACROS 1//Определение макроса препроцессора

#include <stdio.h>//включает заголовочный файл в текущий

Дополнительная информация:

Пример возможных макросов с параметрами:

 #define MAX(a, b) a >= b ? a : b

Здесь не рекомендуется передавать в качестве параметров макроса выражения и вызовы функций:

//Не делайте так!

MAX(++x, --y)

#define SWAP(type, a, b) ({ type tmp = a; a = b; b = tmp; })

//или

#define SWAP(typeof(a), a, b) ({ type tmp = a; a = b; b = tmp; })

Здесь использованы фигурные скобки, это говорит об использовании statement expression,

что позволяет нам задать новую область видимости для переменной tmp и безопасно объявить её внутри макроса:

SWAP(int, x, y);

SWAP(float, x, y);

Макросы также можно записывать в несколько строк, но тогда каждая строка, кроме последней, должна заканчиваться символом '\':

#define SWAP(a, b) ({  \

  decltype(a) tmp = a; \

  a = b;               \

  b = tmp; })

Во избежание багов следует задать свою область видимости для многострочных макросов, для этого принято использовать конструкцию do while:

#define MACRO() do { \

    doSomething(); \

    doSomethingElse(); \

  } while(0)

Параметр макроса можно превратить в строку, добавив перед ним знак '#':

#define PRINT\_VALUE(value) printf("Value of %s is %d", #value, value);

int x = 5;

PRINT\_VALUE(x)  // -> Value of x is 5

А еще параметр можно приклеить к чему-то еще, чтобы получился новый идентификатор. Для этого между параметром и тем, с чем мы его склеиваем, нужно поставить '##':

#define PRINT\_VALUE (number) printf("%d", value\_##number);

int value\_one = 10, value\_two = 20;

PRINT\_VALUE(one)  // -> 10

PRINT\_VALUE(two)  // -> 20

statement expression не является частью стандарта C, она поддерживается компиляторами gcc и clang.

В языке Си при помощи макросов можно эффективно избавляться от дублирования кода. Банальный пример - объявим несколько функций сложения для работы с разными типами данных:

#define DEF\_SUM(type) type sum\_##type (type a, type b) { \

  type result = a + b; \

  return result; \

}

Теперь чтобы получить несколько таких функций для нужных нам типов, нужно просто использовать пару раз этот макрос в глобальной зоне видимости:

DEF\_SUM(int)

DEF\_SUM(float)

DEF\_SUM(double)

int main() {

  sum\_int(1, 2);

  sum\_float(2.4, 6,3);

  sum\_double(1.43434, 2,546656);

}

Однако, с помощью этого макроса не получится сгенерировать функции для типов, состоящих более, чем из 2-х слов.