Лекция 18

Директивы препроцессора. Функции с переменным числом параметров

Преподаватель Палехова Ольга Александровна,
кафедра О7 БГТУ «Военмех»

Препроцессор языка Си

Препроцессор Си (англ. *pre processor*, предобработчик) – программа, подготавливающая код программы на языке Си к компиляции.

Директивы *(команды)* препроцессора

- могут быть в любом месте программы,
- располагаются на отдельных строках,
- начинаются с символа #

Наиболее часто используемые директивы:

- ***#include*** – вставляет в данном месте содержимое указанного файла;
- ***#define*** – производит замены в тексте;
- ***директивы условной компиляции*** – средство для выборочного включения того или иного текста программы в зависимости от значения условия, вычисляемого во время препроцессирования.

Директива *#include*

Любая строка вида

```
#include <имя_файла>
```

или

```
#include "имя_файла"
```

заменяется содержимым файла с именем *имя_файла*.

Особенности:

- `<stdio.h>` - файл находится в стандартной папке *include* или в папке, путь к которой прописан в свойствах проекта;
- `"my_file.c"` – файл находится в папке с исходными файлами программы;
- `"f:\Program\my_func\my_func.h"` – файл находится в указанной папке.

! *#include* позволяет собрать вместе объявления всей программы, чтобы их могли видеть все исходные файлы

Директива макроподстановок *#define*

Простая макроподстановка

```
#define имя замещающий_текст
```

Примеры:

```
#define N 10
#define d_print printf("Отладка: %d\n", x)
```

Далее по тексту

```
x = N;
d_print;
```

лексема

лексема

```
x = 10;
printf("Отладка: %d\n", x);
```

подстановка

подстановка

```
puts ("No");
n = size_N;
metod_print();
```

часть строкового литерала

часть идентификатора

```
puts ("No");
n = size_N;
metod_print();
```

часть идентификатора

Директива макроподстановок *#define*

Макроподстановка с аргументами (макрос) – замещающий текст зависит от задаваемых параметров

Примеры:

без пробела!

скобки обязательны

```
#define max(A,B) ((A) > (B) ? (A) : (B))
#define ABS(A) ((A) >= 0 ? (A) : -(A))
```

Далее по тексту

```
max(a+b, x-y)
ABS(a-b)
```

```
((a+b) > (x-y) ? (a+b) : (x-y))
((a-b) >= 0 ? (a-b) : -(a-b))
```

! **Обращение к макросу выглядит как функция, но это только текстовая замена!**

```
#define N 9
#define q(X,Y) q[(X)*N+(Y)]
```

двойная подстановка

Далее по тексту

```
q(i+1,j) = x;
```

```
q[(i+1)*9+(j)] = x;
```

Директива макроподстановок *#define*

Если в замещающем тексте перед формальным параметром стоит знак #, этот параметр будет заключен в кавычки, при этом возможна конкатенация (склеивание) строк.

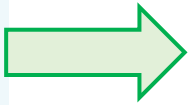
Пример:

заключить параметр в кавычки

```
#define dprint(expr) printf(#expr" = %g\n", expr)
```

Далее по тексту

```
dprint(x);
dprint(sin(x));
dprint(a-b);
```



```
printf("x = %g\n",x);
printf("sin(x) = %g\n",sin(x));
printf("a-b = %g\n",a-b);
```

Действие макроопределения прекращается директивой

#undef имя

Пример:

```
#define sin cos
y = sin(x);
#undef sin
y = sin(x);
```

подстановка выполняется

```
y = cos(x);
```

подстановка отменена

```
y = sin(x);
```

Директивы условной компиляции

Условная компиляция позволяет создавать разные объектные файлы из одного текста программы.

Применение:

- настройка программы под платформу компилятора,
- включение/исключение некоторых фрагментов программы (например, при отладке),
- проверка подключения файла строго один раз.

Директивы условной компиляции:

начало блока условной компиляции

```
#if константное_выражение
#ifdef идентификатор
#ifndef идентификатор
#else
#elif
#endif
```

если идентификатор был объявлен с помощью #define

если с помощью #define такой идентификатор не объявлен

конец блока условной компиляции

Директивы условной компиляции

Проверка включения файла:

```
#ifndef MY_FUNC
   #define MY_FUNC
   #include "my_func.h"
#endif
```

если MY_FUNC не определен

определить MY_FUNC

включить содержимое файла

конец директивы #ifndef

Включение/исключение фрагмента программы

```
#ifdef DEBUG
   puts ("Debug:");
   dprint(x);
   puts ("Array a:");
   for ( i = 0 ; i < N ; i++ )
      printf ("%d ", a[i]);
   printf ("\n");
#endif
```

если включен режим отладки (ранее есть строка #define DEBUG), то фрагмент до #endif включается в компилируемый текст, в противном случае этот фрагмент исключается из текста программы

конец директивы #ifdef

Параметры функции *main()*

Стандартный заголовок функции ***main()*** :

```
int main(int argc, char *argv[])
```

количество аргументов

аргументы

Особенности:

- аргументы передаются программе с помощью командной строки, в которой они разделяются пробелами или табуляциями;
- все аргументы – строки;
- *argc* (сокр. *argument count*) – количество аргументов, задаваемых в командной строке (всегда ≥ 1);
- *argv* (сокр. *argument vector*) – массив символьных строк, содержащих сами аргументы (*argv[0]* – имя вызываемой программы);
- имена параметров можно менять, но нежелательно

Параметры функции *main()*

Задача 1. Выполнить обработку файла, имя которого может передаваться программе через командную строку.

```
#include <stdio.h>
#include <string.h>
int main(int argc, char *argv[])
{
    char filename[80];
    if ( argc > 1 )
        strcpy (filename, argv[1]);
    else
    {
        puts ("Введите имя файла для обработки");
        gets (filename);
    }
    /* обработка файла */
    return 0;
}
```

если аргументы переданы

первый аргумент после имени программы

Параметры функции *main()*

Задача 2. Вычислить и вывести на экран среднее арифметическое значение чисел, передаваемых программе при ее вызове.

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    double sum = 0;
    int i;
    if ( argc > 1 )
    {
        for( i = 1 ; i < argc ; i++ )
            sum += atof (argv[i]);
        printf("%f\n", sum / (argc - 1) );
    }
    else
        printf ("Слагаемые не указаны\n");
    return 0;
}
```

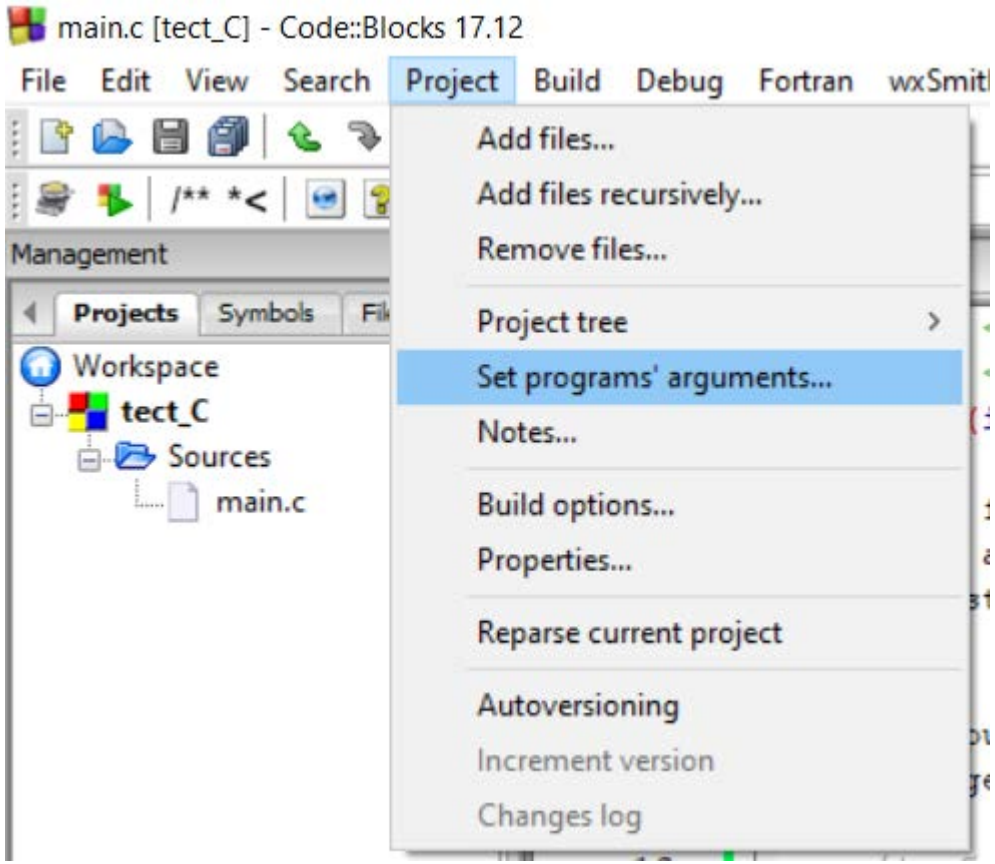
если аргументы переданы

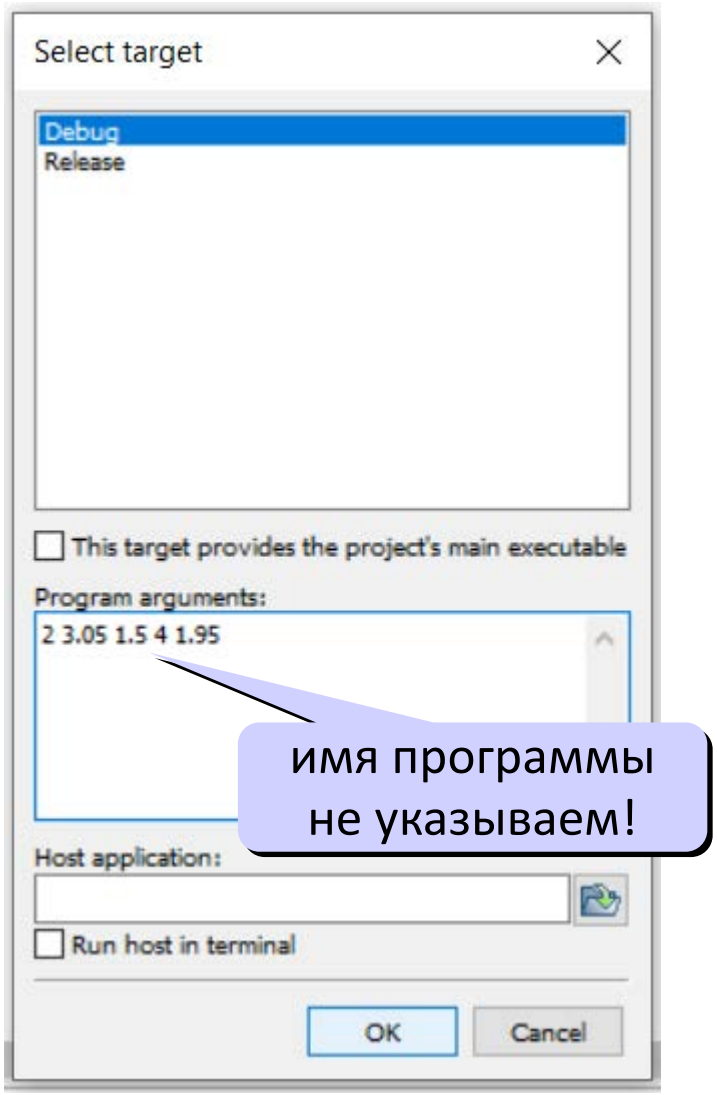
argv[0] – имя программы

каждый аргумент - слово

Параметры функции *main()*

При отладке программы аргументы в функцию main() передаются с помощью пункта *«Program arguments»* меню «Project» (или «Run») IDE.





Параметры функции *main()*

Для Windows и UNIX, как правило, допускается расширение списка параметров:

```
int main(int argc, char *argv[], char *envp[])
```

envp (сокр. *environmental parameters*) – массив символьных строк, представляющих переменные, заданные в среде окружения (среде выполнения программы). Значение последнего элемента массива равно *NULL.*

Пример. Программа, выводящая на экран информацию о переменных окружения

```
#include <stdio.h>
int main(int argc, char *argv[], char *envp[])
{
    int i;
    for ( i = 0 ; envp[i] != NULL ; i++ )
        printf("%s\n", envp[i]);
    return 0;
}
```

конец массива определяется значением NULL

Функции с переменным числом параметров

Если количество параметров функции заранее не известно, используют **функции с переменным числом параметров**

```
тип_рез-та имя_ф-ции (список_обязат_пар-ров, ...);
```

Проблема: как обращаться к безымянным параметрам?

Решение: использовать специальный указатель типа ***va_list***

Проблема: как узнать адрес первого необязательного параметра?

Решение: использовать специальный макрос

```
va_start(va_list_указатель, последний_явный_параметр);
```

Проблема: как узнать значение необязательного параметра?

Решение: использовать специальный макрос

```
va_arg(va_list_указатель, тип_параметра);
```

Требуется **#include <stdarg.h>**

Функции с переменным числом параметров

Задача. Написать функцию для вычисления суммы произвольного количества целочисленных слагаемых

Проблема: как узнать количество переданных слагаемых?

Идея 1: количество неявных параметров передавать в списке обязательных параметров

```
int summa (int count, ...)
{
    va_list p;
    int s = 0;
    va_start(p, count);
    while ( count-- )
        s += va_arg(p, int);
    va_end(p);
    return s;
}
```

количество слагаемых

объявление указателя

устанавливаем указатель на первый необязательный параметр

значение очередного аргумента указанного типа и приращение указателя

тип

завершение работы со списком параметров

Сумма произвольного числа слагаемых

```
#include<stdio.h>
#include<stdarg.h>
int summa(int, ...);
int main(void)
{
    printf("summa = %d\n", summa(2, 5, 7));
    printf("summa = %d\n", summa(5, 1, 2, 3, 4, 5));
    return 0;
}
int summa(int count, ...)
{
    va_list p;
    int s=0;
    va_start(p, count);
    while ( count-- )
        s += va_arg(p, int);
    va_end(p);
    return s;
}
```

прототип функции

количество слагаемых

слагаемые

? Что изменить, чтобы вычислить сумму вещественных чисел?

Сумма произвольного числа слагаемых

```
#include<stdio.h>
#include<stdarg.h>
double summa(int, ...);
int main(void)
{
    printf("summa = %f\n", summa(2, 5.0, 7.0));
    printf("summa = %f\n", summa(5, 1.1, 2., .3, 4.7, 2.5));
    return 0;
}
double summa(int count, ...)
{
    va_list p;
    double s=0;
    va_start(p, count);
    while ( count-- )
        s += va_arg(p, double);
    va_end(p);
    return s;
}
```

вещественные слагаемые

? Что изменилось?

! **Для необязательных параметров преобразование типа не осуществляется!**

Функции с переменным числом параметров

Задача. Написать функцию для вычисления суммы произвольного количества целочисленных слагаемых

Проблема: как узнать количество переданных слагаемых?

Идея 2: последним параметром передавать признак конца последовательности, например 0

```
int summa (int value, ...)
{
    va_list p;
    int s = value;
    va_start(p, value);
    while ( value = va_arg(p, int))
        s += value;
    va_end(p);
    return s;
}
```

первое слагаемое

объявление указателя

первое слагаемое сразу добавляем к сумме

установка указателя

значение очередного аргумента указанного типа и приращение указателя

пока не 0

завершение работы со списком параметров

Сумма произвольного числа слагаемых

```
#include<stdio.h>
#include<stdarg.h>
int summa(int, ...);
int main(void)
{
    printf("summa = %d\n", summa(2, 5, 7, 0));
    printf("summa = %d\n", summa(5, 1, 2, 3, 4, 5, 0));
    return 0;
}
int summa(int value, ...)
{
    int s = value;
    va_list p;
    va_start (p, value);
    while ( value = va_arg (p, int) )
        s += value;
    va_end(p);
    return s;
}
```

прототип функции

слагаемые

признак окончания

Функции с переменным числом параметров

Задача. Написать функцию для вычисления суммы произвольного количества целочисленных и вещественных слагаемых

Проблема: неизвестно не только количество, но и **тип** параметра

Идея: обязательным параметром передавать строку формата, как в функциях *printf()* и *scanf()*

```
double summa (char *, ...);
int main()
{
    int a = 2;
    printf("summa = %f\n", summa("fif", 2., 5, 7.));
    printf("summa = %f\n",
        summa("ffiffi", 3.1, 1.3, a, 3.1, .2*a, 5));
    return 0;
}
```

строка формата

слагаемые

строка формата

слагаемые

Сумма произвольного числа слагаемых

```
double summa(char * st, ...)
{
    double s = 0, f;
    int i;
    va_list p;
    va_start (p, st);
    while (*st)
    {
        if ( *st == 'i' )
        {
            i = va_arg (p, int);
            s += i;
        }
        else
          if ( *st == 'f' )
          {
              f = va_arg(p, double);
              s += f;
          }
          else break;
        st++;
    }
    va_end(p);
    return s;
}
```

до конца строки

если *'i'*, то преобразуем в int

если *'f'*, то преобразуем в double

если не 'i' и не *'f'*, то заканчиваем вычисления