acm.mipt.ru

олимпиады по программированию на Физтехе

Раздел «Язык Си» . CoffeMacros :

Команды препроцессора

Тут описать цепочку препроцессор - компилятор - линкер.

Команды препроцессора начинаются со знака #

Команда препроцессора #define

Поиск и замена.

```
#define N 1000
```

Дальше по файлу ищется идентификатор (N) и заменяется на все, что стоит после идентификатора (1000)

Удобно использовать для задания размеров массивов (если в задаче размер массива будет не 10, а 100, мы поменяем 10 на 100 только в одном месте.

```
#include <stdio.h>

#define N 10

int main()
{
    int i, a[N];    // после работы препроцессора заменится на int i, a[10];

    for(i=0; i<N; i++)        // это for(i=0; i<10; i++)
        scanf("%d", &a[i]);

    for(i=0; i<N; i++)       // это for(i=0; i<10; i++)
        printf("%d ", a[i]);
    for(i=0; i<N; i++)       // это for(i=0; i<10; i++)
        printf("%d ", a[i]);
    printf("\n");
    return 0;
}
```

НЕ ставьте ; (точку с запятой) в конце макроса

```
#include <stdio.h>

// поставим в конце макроса ;
#define N 10 ;

int main()
{
    int i, a[N];    // после работы препроцессора заменится на int i, a[10;]; ОШИБКА!

    for(i=0; i<N; i++)        // это for(i=0; i<10; ; i++)  ОШИБКА
        scanf("%d", &a[i]);
```


Поиск

Поиск

Раздел «Язык Си»

Главная
Зачем учить С?
Определения
Инструменты:
Поиск
Изменения
Index
Статистика

Разделы

Информация
Алгоритмы
Язык Си
Язык Ruby
Язык Ассемблера
El Judge
Парадигмы
Образование
Сети
Objective C

Logon>>

```
    // еще код
    return 0;
}
```

Добавим аргументы

Хотим написать функцию, которая считает минимум из 2 чисел.
Какой тип аргументов и возвращаемого значения должен быть у этой функции?

int min(int x, int y); сравнит только целые числа и не сравнит дробные, long long переменные тоже не сравнит. Надо будет написать много функций – для сравнения каждого типа (можем упростить задачу и написать для long long и double, но получим много дополнительного кода при вызове).

```
#include <stdio.h>

int min(int x, int y)
{
    if (x<y)
        return x;
    else
        return y;
}
// или с помощью оператора ? :
int min1(int x, int y)
{
    return (x<y) ? x : y;
}

int main()
{
    printf("%d\n", min(3, 5));  // 3
    printf("%d\n", min(3, -5)); // -5
    printf("%d\n", min(5, 5));  // 5

    return 0;
}
```

Как написать min для любого типа?

```
#include <stdio.h>

#define min(x,y) x < y ? x : y

int main()
{
    printf("%d\n", min(3, 5));      // 3
    printf("%d\n", min(3, -5.5));   // -5.5
    printf("%d\n", min(5.1, 5.2));  // 5.1

    return 0;
}
```

Опять поиск и замена. Берется выражение min(3,5) и заменяется выражением. В выражении x заменяется на 3, y заменяется на 5. (Имена аргументов можно писать любые.)

 Тип аргументов и возвращаемого значения не указывается.

Добавьте скобки

Чему равно значение выражения

```
min(3, 5)+2
```

Это выражение заменится на

```
3<5 : 3 ? 5 + 2
```

Т.е. значение выражения 3. Как исправить? Надо поставить скобки вокруг выражения.

```
#define min(x,y) (x < y ? x : y)
```

Достаточно ли поставить скобки вокруг любой реализации макроса, чтобы он однозначно работал?

Больше скобок

Напишем макрос, считающий квадрат числа.

```
#define kvadrat(x) (x*x)
```

Чему равно значение выражения?

```
kvadrat(3+2)
```

Это выражение заменится на

```
(3+2*3+2)
```

Получится 11, а не 25, как мы ожидали. Что делать? Поставить еще больше скобок при написании макроса.

```
#define kvadrat(x) ((x)*(x))
```

Скобки вокруг всего макроса и вокруг его аргументов - этого достаточно?

Как нельзя использовать макросы (когда скобки не помогают)

Макрос нужно не только хорошо написать, но и правильно вызвать.

Используем макрос kvadrat. Что будет напечатано?

```
int x = 2;
int y = kvadrat(x++);
printf("y=%d x=%d\n", y, x);
```

Может быть напечатано два любых числа. Подробнее об этом в FAQ, вопросы 4.1-4.4.

Рекомендация: не используйте операторы ++ и -- при вызове макросов.

Более общая рекомендация: пишите в параметрах макроса только то, что может быть выполнено много раз подряд без изменения результата.

Чем отличается макрос от функции?

- в функции есть типы аргументов и тип возвращаемого значения;
- макрос - это замена одного текста программы на другой текст программы, операторы этого текста выполняются один за другим; функция - это передача управления в функцию и возврат из нее + передача параметров.
- кто быстрее?
- время компиляции и время исполнения кода.
- не все выражения безопасно указывать в виде аргументов макроса.

Напишем проверку является ли число ASCII-кодом цифры или нет в виде макроса isdigitM и функции isdigitF и используем их.

```
#define <stdio.h>

int isdigitF (int x)
```

```
{
    return '0'<= x && x <= '9';
}
int main()
{
    int c;
    while(isdigitF(c=getchar()))
        putchar(c);
    return 0;
}
```

Введем символы 12345 и нажмем Enter.

На экране напечатается 12345.

Напишем ту же проверку через макрос isdigitM

```
#define <stdio.h>

#define isdigitM ('0'<= (x) && (x) <= '9')

int main()
{
    int c;
    while(isdigitM(c=getchar()))
        putchar(c);
    return 0;
}
```

Введем символы 12345 и нажмем Enter.

На экране напечатается 24.

В чем разница? При вызове функции сначала считается значение аргумента при вызове (один раз на аргумент), то есть *один* раз на каждую итерацию цикла вызывается код c=getchar(). Потом вызывается функция с аргументом, равным полученному ASCII-коду.

При работе макроса препроцессор заменяет функцию main на следующий код:

```
int main()
{
    int c;
    while(('0'<= (c=getchar()) && (c=getchar()) <= '9'))
        putchar(c);
    return 0;
}
```

То есть c=getchar() выполняется **два раза** при каждой итерации цикла. Первый раз считывается символ 1 и сохраняется в переменную c, второй раз - символ 2 и записывается в ту же переменную (затирая предыдущее значение). Печатает putchar символ, код которого хранится в c. Это 2. Символ 1 не будет напечатан. Аналогично для последовательности символов 3 и 4. Прочитаются два символа (два вызова getchar), а напечатается только один символ (один вызов putchar).

Вывод: макросы удобны, но нужно уметь их писать и использовать.

__LINE__, __FUNCTION__, __FILE__ и другие

макрос	значение
__LINE__	номер строки в файле
__FUNCTION__	имя функции, внутри которой написан __FUNCTION__
__FILE__	имя файла, в котором написан этот макрос
__TIME__	время компиляции файла

Как использовать?

Для отладочной печати удобно воспользоваться макросом, который печатает номер строки и функцию.

```
#include <stdio.h>

#define prn (fprintf(stderr, "%d : %s\n", __LINE__, __FUNCTION__))

int main()
{
    int i;

    prn;
    for(i = 0; i < 5; i++);
    {
        prn;
        printf("i=%d\n", i);
    }
    prn;
    return 0;
}
```

При входных данных

```
1 2 3 4 5 6 7 8 9 10
```

печатается

```
9 : main
12 : main
i=0
12 : main
i=1
12 : main
i=2
12 : main
i=3
12 : main
i=4
15 : main
```

Условная компиляция

Директивы #, ## и другие

-- TatyanaDerbysheva - 24 Oct 2016