#ifndef ПРЕПРОЦЕССОР #define ПРЕПРОЦЕССОР #endif

Workshop by

Danil Borchevkin

danil@borchevkin.com

+79114692217

Resources

Книги

- ISBN 978-5-8459-0891-9 Керниган, Ритчи «Язык программирования Си». Глава 4.11, Приложение A.12
- ISBN 978-5-9775-0145-3 Полубенцева «C/C++ Процедурное программирование». Глава 5.
- ISBN 978-5-8459-0986-2 Прата «Язык программирования C. Лекции и упражнения». Глава 16
- https://tproger.ru/translations/c-macro/

Время

- ??? Часов на подготовку
- ???

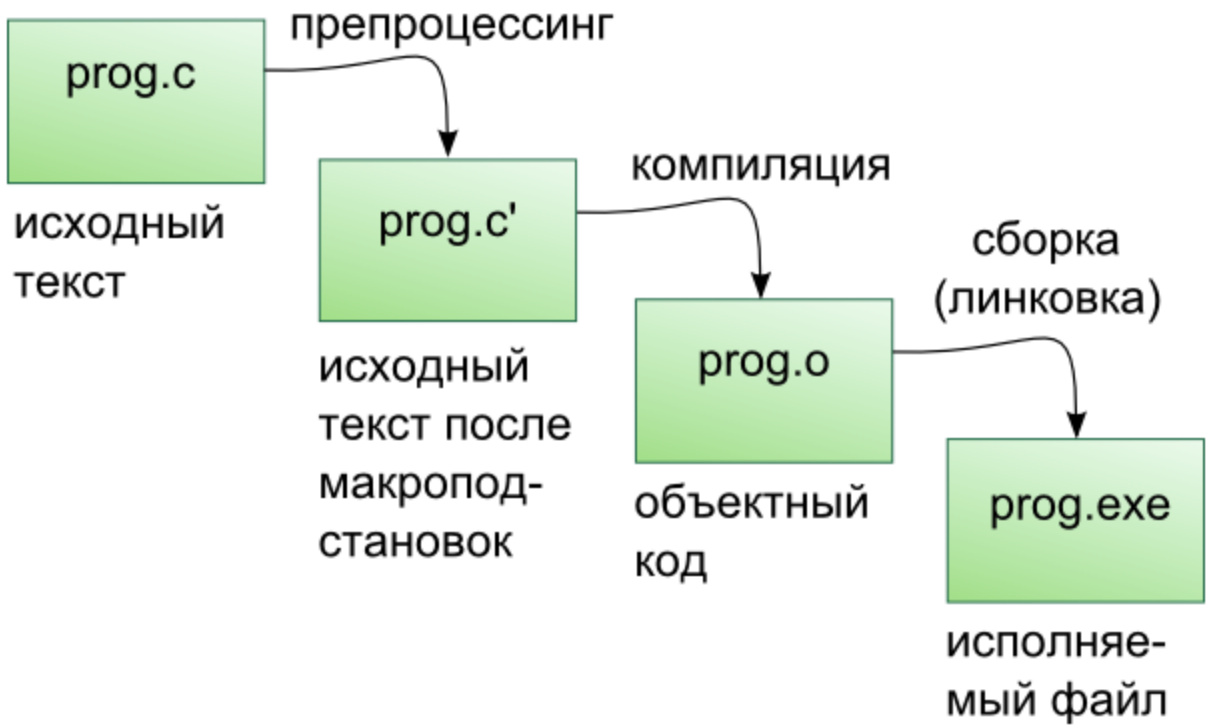
Ресурсы

- 2 бутылки Grinbergen

Место препроцессора в языке Си

На этапе прецпроцессинга производятся:

- Замена комментариев пустыми строками
- Макроподстановки
- Обработка директив условной компиляции
- Текстовое включение файлов



Список директив препроцессора

Директива	Пояснение
#include	Включает содержимое других файлов
#define	Задает макроподстановку
#undef	Отменяет действие #define
#if	Директивы условной трансляции позволяют включать и исключать из файла куски кода на стадии препроцессора
#elif	
#else	
#endif	
#ifdef	
#ifndef	
#pragma	Настраивает компилятор с учетом специфики. Отличается для разных компиляторов
#line	Включает номера строк исходного кода заимствованных файлов в диагностику
#error	Вывод сообщения компилятором и остановка компиляции

Макропостановка

```
#define MULT 5
#define MULTIPLY(x,y) ((x)*(y)*MULT)
```

Простое макроопределение

```
#define <идентификатор макроса> <тело макроса>
```

```
#define TRUE        1
...
if(n == TRUE) {
        ...
}
```

Важно помнить, что препроцессор просматривает текст несколько раз. После первой подстановки снова выполняется просмотр уже расширенного текса.
Это позволяет организовывать вложенные макросы.

```
#define TRUE        1
#define FALSE       !TRUE
```

Макрос с параметрами

```
#define <идентификатор макроса>(параметры) <тело макроса>
```

Некоторые вещи нельзя написать с помощью макроса:

```
#define SQUARE(x) ((x)*(x))
int z = 2, y;

y = SQUARE(z++)
```

Важно:

- **Нельзя ставить пробел перед открывающей скобкой параметров**
- Тело макроса должно учитывать возможный порядок выполнения операций после подстановки. Используйте скобки

```
#define SQUARE_WRONG(x) x*x
#define SQUARE_RIGHT(x) ((x)*(x))
```

Многострочный макрос

```
#define <идентификатор макроса>(параметры) <тело макроса> \
                                    <тело макроса> \
                                    <тело макроса>
```

```
#define MAKE_32(b1, b2, b3, b4) ( (uint8_t)b1  | \
(uint8_t)b2 << 8 | \
(uint8_t)b3 << 16 | \
(uint8_t)b4 << 24)
```

Пустой макрос

```
#define <идентификатор макроса>
```

- Встречая пустой макрос, препроцессор просто исключает его из тела программы.
- В основном пустой макрос используется для условной компиляции кода
- Одно из крутых применений – определение макроса при компиляции – ключ -**D** (gcc)

Конкатенация макросов

```
#define NUM(name,number) name##_##number
```

- Сочетание ## сливает несколько лексем в одну
- Встречал крайне редко, кроме явного указания типов

```
#define _TEXT(x) L##x
```

Предопределенные маросы

Директива	Пояснение
__DATE__	**Строка** с датой компиляции
__FILE__	**Строка** с именем файла
__FUNCTION__	**Строка** с именем функции
__LINE__	**Число** с номером текущей строчки
__TIME__	**Строка** с временем компиляции
и т.д.	См. документацию на свой компилятор

Самая ебнутая директива

#line <новый номер строки> “имя файла”

- Директива используется для дебага и отладки.
- *<новый номер строки>* - переопределяет номер следующей строки (переопредетяет макрос __LINE__)
- *“имя файла”* – необязательный параметр. Переопределяет макрос __FILE__

```
#include <stdio.h>
#line 100
int main(void)  {
  printf ("%d\n",__LINE__);
  return 0;
}
```

Снятие определения макроса

```
#undef <идентификатор макроса>
```

- Выполняется корректно, вне зависимости, определен макрос или нет.
- В **Си один макрос нельзя переопределять несколько раз**. Это вызывает ошибку. Используй сначала #undef

Условная компиляция

```
#ifdef MSP430
…
#else
#error “No MSP430 defined. Abort compiling”
```

#ifdef, ifndef, #else, #endif

- Мы добрались до использования пустого макроса!
- Довольно популярная конструкция.
- Часто используется для избегания ошибок, связанных с переопределением или повторным включением заголовочных файлов

```
#ifdef <идентификатор макро>
   …
#else
   …
#endif
```

#if, #elif, #else, #endif, оператор defined

- Более гибкие, чем предыдущие собратья
- В #if можно использовать сложные условия
- #if defined MACRO ~ #ifdef MACRO
- #if !defined MACRO ~ #ifndef MACRO

```
#if defined MACRO && MACRO >5
   …
#endif
```

```
#if <условие>
   …
#elif <условие>
   …
#elif <условие>
   …
#else
   …
#endif
```

#error

- Полезная штука, если вам нужно использовать большое количество макросов, а от их правильного определения зависит работа программы
- Достижение #error приостанавливает компиляцию и выводит ошибку компилятора

```
#ifndef <идентификатор макро>
  #error “Gotcha!”
#endif
```

Включение именованных файлов

#include <stdint.h>

#include “bsp.h”

Включение именованных файлов

```
#include “bsp.h”
#include <string.h>
```

- Если подключаемый файл указан в <>, то поиск будет происходить в стандартных каталогах, предназначенных для хранения заголовочных файлов.
- В случае, если подключаемый файл заключен в двойные кавычки, поиск будет происходить в текущем рабочем каталоге. Если файл не найден, то поиск продолжается в стандартных каталогах.c

```
// Guardian
#ifndef _BSP_H
#define _BSP_H

...

#endif // _BSP_H
```

Danger Zone!

Проблемы макросов и пути их избегания

Мнения о макросах

- «Злоупотребение макросами чревато возникновением трудно выявляемых ошибок» - Страуструп
- «Оставим макроопределение профессионалам, а сами будем использовать созданное профессионалами» - Полубенцева
- «если ты только точно не знаешь что делаешь - не используй макросы» - неизвестный пользователь cyberforum
- «… и вообще, это дурной тон» - неизвестный пользователь cyberforum

Проблемы макросов

- на стадии их "расширения" не производится проверка компилятором. потому, отыскать ошибки, связанные с макросами, крайне сложно.
- Соответственно, нет проверки типов.
- Написание макросов с параметрами требует определенного скилла, а ошибки порождаемые использованием макросов, трудно отлаживаемые.
- Макросы с параметрами имеют ограниченное применение (к примеру, невозможность использования в некоторых макросах переменной с инкрементом).
- Ухудшается читаемость кода.

Плюсы макросов

- Нет затрат, связанных с вызовом функции (затраты на создание кадра стека). Соответственно выполнение макроса с параметрами быстрее, чем вызов функции.
- Можно расширять синтаксис языка (смотри FOREACH)

```
#define foreach(item, array) \
    for(int keep = 1, \
            count = 0,\
            size = sizeof (array) / sizeof *(array); \
        keep && count != size; \
        keep = !keep, count++) \
      for(item = (array) + count; keep; keep = !keep)

     int main(void)
{
    int values[] = { 1, 2, 3 };


    foreach(int *v, values)
        printf("value: %d\n", *v);


    return 0;
}
```

Если не макрос, то…

- необходимость в макросах с параметрами отпала с появлением inline-функций, которые есть и в C++, и в C99
- #define лучше заменять константой или перечислением, где это возможно

Вместо макроса	Используйте
Определение константы #define YES 0	const uint8_t YES = 0; enum {YES, NO};
Избежать расходов на вызов функции #define SQUARE(x) ((x)*(x))	inline uint16_t SQUARE(uint8_t x) {return x*x;}

#ifndef ПРЕПРОЦЕССОР
#define ПРЕПРОЦЕССОР
#endif

Danil Borchevkin

danil@borchevkin.com

+79114692217