Директивы препроцессора.

Директивы препроцессора

- Макроопределения
  - #define, #undef
- Директива включения файлов
  - #include
- Директивы условной компиляции
  - #if, #ifdef, #endif и др.

Остальные директивы (#pragma, #error, #line и др.) используются реже.

Правила, справедливые для всех директив

- Директивы всегда начинаются с символа "#".
- Любое количество пробельных символов может разделять лексемы в директиве.
- Директива заканчивается на символе '\n'.
- Директивы могут появляться в любом месте программы.

Правила, справедливые для всех директив (пояснения)

- Любое количество пробельных символов могут разделять лексемы в директиве.

```
#  define   N   1000
```

- Директива заканчивается на символе '\n'.

```
#define DISK_CAPACITY   (SIDES *                  \
                         TRACKS_PER_SIDE *        \
                         SECTORS_PER_TRACK *      \
                         BYTES_PER_SECTOR)
```

Простые макросы

#define идентификатор список-замены

```
#define PI 3.14
#define EOS '\0'
#define MEM_ERR "Memory allocation error."
```

Используются:

- В качестве имен для числовых, символьных и строковых констант.

Продолжение на следующем слайде.

Простые макросы

Окончание предыдущего слайда.

- Незначительного изменения синтаксиса языка.

```
#define BEGIN {
#define END }
#define INF_LOOP for( ; ; )
```

- Переименования типов.

```
#define BOOL int
```

- Управления условной компиляцией.

Макросы с параметрами

#define идентификатор(x1, x2, ..., xn) список-замены

- Не должно быть пробела между именем макроса и (.
- Список параметров может быть пустым.

```
#define MAX(x, y) ((x) > (y) ? (x) : (y))
#define IS_EVEN(x) ((x) % 2 == 0)
```

Где-то в программе

```
i = MAX(j + k, m - n);
// i = ((j + k) > (m - n) ? (j + k) : (m - n));

if (IS_EVEN(i))
// if (((i) % 2 == 0))
    i++;
```

Макросы с переменным числом параметров (C99)

```
#ifndef NDEBUG
#define DBG_PRINT(s, ...) printf(s, __VA_ARGS__)
#else
#define DBG_PRINT(s, ...) ((void) 0)
#endif
```

Макросы с параметрами vs функции

Преимущества

- программа может работать немного быстрее;
- макросы "универсальны".

Недостатки

- скомпилированный код становится больше;

```
n = MAX(i, MAX(j, k)));
```

- типы аргументов не проверяются;
- нельзя объявить указатель на макрос;
- макрос может вычислять аргументы несколько раз.

```
n = MAX(i++, j);
```

Общие свойства макросов

- Список-замены макроса может содержать другие макросы.
- Препроцессор заменяет только целые лексемы, не их части.
- Определение макроса остается «известным» до конца файла, в котором этот макрос объявляется.
- Макрос не может быть объявлен дважды, если эти объявление не тождественны.
- Макрос может быть «разопределен» с помощью директивы #undef.

Скобки в макросах

- Если список-замены содержит операции, он должен быть заключен в скобки.
- Если у макроса есть параметры, они должны быть заключены в скобки в списке-замены.

```
#define TWO_PI  2 * 3.14

f = 360.0 / TWO_PI;
// f = 360.0 / 2 * 3.14;

#define SCALE(x)  (x * 10)

j = SCALE(i + 1);
// j = (i + 1 * 10);
```

Создание длинных макросов

```
// 1
#define ECHO(s)     {gets(s); puts(s);}

if (echo_flag)
    ECHO(str);
else
    gets(str);

// 2
#define ECHO(s)  (gets(s), puts(s))

ECHO(str);
```

Создание длинных макросов

```
#define ECHO(s) \
do              \
{               \
    gets(s);    \
    puts(s);    \
}               \
while(0)        \
```

Предопределенные макросы

- **__LINE__** - номер текущей строки (десятичная константа)
- **__FILE__** - имя компилируемого файла
- **__DATE__** - дата компиляции
- **__TIME__** - время компиляции

- **и др.**

Эти идентификаторы нельзя переопределять или отменять директивой undef.

- **__func__** - имя функции как строки (GCC only, C99 и не макрос)

Условная компиляция

Использование условной компиляции:

- программа, которая должна работать под несколькими операционными системами;
- программа, которая должна собираться различными компиляторами;
- начальное значение макросов;
- временное выключение кода.

Условная компиляция

```
#if defined(OS_WIN)
...
#elif defined(OS_LIN)
...
#elif defined(OS_MAC)
...
#endif
```

```
#ifndef BUF_SIZE
#define BUF_SIZE  256
#endif
```

```
#if 0
for(int i = 0; i < n; i++)
    a[i] = 0.0;
#endif
```

Остальные директивы

#error сообщение

```
#if defined(OS_WIN)
...
#elif defined(OS_LIN)
...
#elif defined(OS_MAC)
...
#else
#error Unsupported OS!
#endif
```

Директива #pragma позволяет добиться от компилятора специфичного поведения.

«Операция»

«Операция» # конвертирует аргумент макроса в строковый литерал.

```
#define PRINT_INT(n) printf(#n " = %d\n", n)

#define TEST(condition, ...) ((condition) ?        \
        printf("Passed test %s\n", #condition) :  \
                                  printf(__VA_ARGS__))
```

Где-то в программе

```
PRINT_INT(i / j);
// printf("i/j" " = %d", i/j);

TEST(voltage <= max_voltage,
      "Voltage %d exceed %d", voltage, max_voltage);
```

«Операция»

«Операция» ## объединяет две лексемы в одну.

```
#define MK_ID(n)    i##n
```

Где-то в программе

```
int MK_ID(1), MK_ID(2);
// int i1, i2;
```

Более содержательный пример

```
#define GENERAL_MAX(type)          \
type type##_max(type x, type y)    \
{                                  \
    return x > y ? x : y;          \
}
```

inline-функции (C99)

inline – *пожелание* компилятору заменить вызовы функции последовательной вставкой кода самой функции.

```
inline double average(double a, double b)
{
    return (a + b) / 2;
}
```

inline-функции по-другому называют встраиваемыми или подставляемыми.

inline-функции (C99)

В C99 inline означает, что определение функции предоставляется только для подстановки и где-то в программе должно быть другое такое же определение этой же функции.

```
inline int add(int a, int b) {return a + b;}

int main(void)
{
    int i = add(4, 5);

    return i;
}
// main.c:(.text+0x1e): undefined reference to `add'
// collect2.exe: error: ld returned 1 exit status
```

Способы исправления проблемы «unresolved reference»

- Использовать ключевое слово static

```
static inline int add(int a, int b) {return a + b;}

int main(void)
{
    int i = add(4, 5);

    return i;
}
```

Способы исправления проблемы «unresolved reference»

- Убрать ключевое слово inline из определения функции.

```
int add(int a, int b) {return a + b;}

int main(void)
{
    int i = add(4, 5);

    return i;
}
```

Компилятор «умный» :), сам разберется.

Способы исправления проблемы «unresolved reference»

- Добавить еще одно **такое же не-inline** определение функции **где-нибудь** в программе.