

01.03.

Директивы препроцессора.

Семинар

1

начало → символ # и обрабатываются во время 1 фазы компиляции.

1) # (препроцесс. обработка)

2) синтаксис. ошибки;

3) перевод в машинный код

Х осн. возможностями препроцессора относятся:

- макро-подстановка;
- включение файлов;
- условная компиляция;

① Точечное макро-подстановки.

Опред. макро-подстановки имеет вид:

#define имя замещающий текст

Макро-подстановка используется для простейшей замены. Во всех местах, где встрет. имя вместо него будет помещён замещающий текст. Имена в макро-подстановке задаются по тем же правилам, что и имена обычных переменных. Замещающий текст может быть произвольным.


```
#define PI 3.141592654
```

```
void main (void)
```

```
{ printf ("%f\n", PI);
```

```
}
```

Обычно зашифрованный текст находится в строке, в которой расположено слово `#define`, но в некоторых определениях его можно продвигать на следующую строку, поставив в конце каждой продвинутой строки `\`.

```
#define matr for(i=0; i<n; i++)\
```

```
for(j=0; j<m; j++)
```

Область действия имени в макро-подстановке простирается от данного определения до конца файла. В определении макро-подстановки могут фигурировать более ранние `#define` определения.

```
#define A PI
```

```
void main (void)
```

```
{ printf ("%f\n", A);
```

```
printf ("%f\n", PI); }
```


Подстановка осуществляется только для тех имен, которые расположены вне текстовых записей в «».

```
printf("PI");
```

Макроподстановку можно определить с аргументами, в результате чего заменяемый текст будет варьироваться в зависимости от задаваемых параметров.

```
#define max(A,B) ((A)>(B)?(A):(B))
```

Хотя обращение к `max` выглядит как обращение к функции, они будут вызывать только текстовую замену. Каждый сформированный параметр будет заменяться соответствующим аргументом.

```
x = max(a+b, c+d);
```

↓

будет заменена на строку

```
x = ((a+b)>(c+d)?(a+b):(c+d));
```

Указанная конструкция предпочтительнее, чем использование функций, т.к. определение

max подходит для данных любого типа.

Тем не менее это и подобные ему текстовые решения следует использовать очень осторожно. Например, размыкание скобок может вызвать исповз. инкрементные операций.

```
x=max(i++,j++);
```

То, что было в `define`, можно скрыть от препроцессора с помощью директивы

`#undef` и имя. Так правильно, это делается,

чтобы перекрыть макро-определение какой-нибудь функцией с этим именем.

```
#define vasia A
```

```
vasia () { ... }
```

```
void main (void)
```

```
{ int vasia;
```

```
;
```

```
#undef vasia;
```

```
vasia ();
```

```
}
```


Имена форм. параметров не заменяются, если они находятся в строках, заключенных в "». Но если в заменяющемся тексте встречается комбинация из форм. параметра и предшествующего ему #, то такая комбинация в макро-расширении будет заменена на аргумент, заключенный в "». Такую конструкцию можно использовать для конкатенации с другими символическими строками.

```
#define print(expr) printf(#expr "=%f\n", expr)
```

```
print(x/y);
```

↓

```
printf("x/y" "=%f\n", x/y);
```

```
printf("x/y=%f\n", x/y);
```

Внутри форм. аргумента каждый знак "» заменяется на \", а \→\", так, чтобы результатом обработки приводил к правильной символической константе.

⑤ Включение файлов

Директива `#include` предписывает компилятору вставить на её место содержимое другого файла. Обычно эта директива применяется к так называемым включенным (заголовочным файлам; *.h). Однако, существует возможность включить в текст программы любой другой исходный файл.

`#include <имя.файла>`

`#include "имя.файла"`

Если имя файла не является полным именем (полный путь до файла), то в 1-ом случае поиск файла происходит только в пределах специфизированных каталогов, включаемых при использовании второй формы сначала просматривается текущий каталог, а затем каталоги включаемых файлов.

③. Условная компиляция

В языке C существует возможность условно компилировать части файла.

В зависимости от значения некоторого константного выражения или идентификатора. Для этой цели служат директивы `#if` (выражение - 1)

текст - 1

`#elif` (выражение - 2)

текст - 2

`#else`

текст - 3

`#endif`

Каждая из этих директив записывается на отдельной строке. Константное выражение в `#if` и последующих `#elif` строках выписывается по порядку, пока не обнаружится выражение с не нулевым значением. Текст, следующий за строкой с нулевым значением, выстраивается. Текст, расположенный за директивой с ненулевым знач., обработ. обычным образом. Под словом "текст" имеется в виду любая послед. строк, выходящая строки пре-

процессора, которые не являются частью
условной структуры. Текст может быть и
пустым. Если `#if` или `#elif` строка с
ненулевым значением выражения найдено,
и ее текст обработан, то после `#elif` и
`#else` строки со своим текстом выбро-
сается. Если все выражения имеют нулевое
значение и присутствует строка `#else`, то
следующий за ней текст обрабатывается
обычным образом.

```
#if system == BCD
    #include "bcd.h"
#elif system == MSDOS
    #include "msdos.h"
#else
    #include "default.h"
#endif
```

Конструкция `#if` можно использовать в
выражении `defined(имя)`, которое дает единицу,
если имя было определено, и 0 - в противном случае.

В свою очередь, директивы `#ifdef` и `#ifndef` эквивалентны `#ifdefined` и `#if!defined`.

Углубление:

```
#ifdef DOS
```

```
#ifndef NDEBUG
```

```
puts(msg);
```

```
#endif
```

```
#endif
```

```
#if!defined(DOS)&&!defined  
(NDEBUG)
```

```
puts(msg);
```

```
#endif
```

04.03. 1.5) Плате Житов

Плате Житов - это набор переменных, которые хранятся совместно в общей области памяти, совпадающей

ния. При вызове в левой части как бы
"привлекается" функция.

Хранение информации в
двоичном коде.

15.03

Семин

При помощи ранее изученных функций (`fscanf`,²
`fgetc`,^(c) `fputs` и проч.) можно организовать

работу с текстовыми файлами. Но наряду
с ними можно исп. и двоичные. Хранение

инфы в двоичных файлах позволяет
существенно ускорить время считы-
вания и записи. Существует 2 способа
доступа к элементам двоичных файлов:

1. последовательный;
2. произвольный.

1) Этот способ используется, как правило,
для перебора всех данных, хранящихся
в файле. Заполнение пустых файлов →
в последовательном режиме. Для откры-
тия и закрытия двоичных файлов

иск. известные функции → fopen, fclose.

Но ко всем решениям доступна функция

"b". Для записи в файл используется

fwrite, её прототип → в stdio.h. Заголовоч

функции в обьекте буге: size_fwrite(const

void *ptr, size_t size, size_t n, FILE* stream).

↑
указатель на
исх. данные,
записываемые
в файл

↑
размер в
байтах
одного
элемент данных

↑
число
записываемых
в файл
эл. данных,
размером
size-байтов
каждый

↑
указатель
на файл-
овый
поток,
открытый
в глобальной
переменной

```
void main(void)
```

```
{ FILE *giena;
```

```
int mass[100];
```

```
giena = fopen("slon.dat", "wb");
```

```
for (i = 0; i < 100; i++)
```

```
    mass[i] = i * i;
```

```
    fwrite(mass, 1 элемент sizeof(int), 100, giena);
```

```
fclose(giena);
```

Тип данных из файла → fread. В отличие
от fwrite её первым аргументом

является адрес переменной, в которую
нужно положить считываемые данные.

Чтобы узнать размер считываемого
файла \rightarrow `filelength`, которая принимается в
качестве аргумента дескриптор файла.

Дескриптор - уникальный код, используемый
для идент. ког.

Для получения дескриптора используется
функция преобразования файлового потока
в дескриптор `fileno`.

```
void main (void)
```

```
{ FILE * giena;
```

```
int desk;
```

```
float x, max;
```

```
giena = fopen("lev.dat", "r+b");
```

```
desk = (номер)fileno(giena);
```

```
fread(&max, sizeof(float), 1, giena);
```

```
for (int i = 1; i < filelength(desk); i++)
```

```
{ fread(&x, sizeof(float), 1, giena);
```

```
if (x > max)
```

```
max = x;
```

```
printf("Максимум = %f", max);
```

```
fclose(giena);
```


2) Организация произвольного доступа к компонентам файла позволяет считывать значения из любой позиции файла, а также записывать новую информацию в любое место файла. Все компоненты файлов с произвольным доступом должны иметь одинаковую длину. Для изменения места файла, откуда будет проводиться следующая операция чтения/записи служит функция `fseek`. Ее прототип содержится в файле `stdio.h`, а заголовок имеет вид:

```
int fseek(FILE* stream, long offset, int whence)
```

Функция возвращает 0 в случае успешного завершения и ненулевое значение в противном случае. Ее параметры:

`FILE* stream` - указатель на откр. файл.

`whence`

(← обратно)

`long offset` - число байтов, на которое нужно переместить файловый указатель.

int whence - указатель на положение точки отсчета, от которого будет происходить перемещение файлового указателя.

- 1) SEEK_SET - перемещение относительно начала файла;
- 2) SEEK_END - относительно конца;
- 3) SEEK_CUR - относительно текущей позиции файлового указателя.

```
void main(void)
```

```
{ FILE *giena;
```

```
int x;
```

```
giena = fopen("slon.dat", "rb");
```

```
fseek(giena, 25 * sizeof(int), SEEK_SET);
```

```
fread(&x, sizeof(int), 1, gienna);
```

```
printf("Прочитано 25 число в файле. Оно равно %d\n", x);
```

```
fclose(gienna);
```

```
}
```

Для определения конца файлового потока →
feof, которая возвращает ненулевую, если

Внутренний указатель файлового потока находится за последним байтом файла, и 0 - в противном случае. На вход функции подается указатель на откр. файл. поток

```
while (!feof(gfena));
```

03.21 2.3 Встроенные функции.

В C++ существует возможность объявить функцию встроенной. Для этого необходимо в заголовке и в прототипе функции вставить `inline`. Такая функция (ее тело) будет вычитаться в программе.

22.03

Стандартные библиотеки языка C.

Семинар 1). h (подключаются в период препроц. обработки на этапе компиляции).

2) $.lib$ (библиотеки, подключаемые на этапе компоновки).

$.obj$
 \downarrow
 $.exe$ ① Математические функции: $\langle math.h \rangle$

1) $\sin x$

$\text{double sin}(\text{double } x)$

$a = \sin(x);$

2) $\cos x$

$\text{double cos}(\text{double } x)$

3) $\text{tg } x$

$\text{double tan}(\text{double } x)$

4) $\arcsin x$

$\text{double asin}(\text{double } x)$

5) $\arccos x$

$\text{double acos}(\text{double } x)$

6) $\text{arctg } x$

$\text{double atan}(\text{double } x)$

Значение, возвращаемое функцией, в том случае, когда аргумент выходит за

пределы области определения, зависят от реализации \rightarrow ошибка области.

7) e^x

`double exp(double x)`

8) $\ln x (x > 0)$

`double log(double x)`

9) $\lg x (x > 0)$

`double log10(double x)`

10) Возведение в степень x^y

`double pow(double x, double y)`

`t = pow(x, y);`

Если $x = 0, y \leq 0$ или $x < 0, y$ - нецелое \Rightarrow

ошибка области.

11) \sqrt{x} (квадратный)

`double sqrt(double x)`

12) Округление с избытком

`double ceil(double x)`

Возвращает наименьшее целое в виде

`double`, которое $\geq x$.

13) Округление с недостатком

double floor(x)

Возвращает наибольшее целое в виде
данного, которое $\leq x$.

14) Модуль (для целых)

int abs(int x)

(для вещественных)

double fabs(double x)

(для длинного целого)

long int labs(long int x)

②. Функции обработки строк <string.h>.

1) char *strcpy(char *s, const char *t)

(Копируем строку t в строку s, включая

\0. Возвращаем строку s).

char a[10], b[10], c[10];

a = strcpy(b, c);

2) char *strcat(const char *s, const char *t)

(Прибавляем t к s и возвращаем

s) char *strchr(const char *s, int c)

(Возвращаем указатель на первое
вхождение символа c в строку s).

Если макрос не определен \Rightarrow NULL.

3) char *strchr (const char *s, int c

(Получает адрес первого вхождения символа c в s).

4) int strlen (const char *s)

(Длина строки) \rightarrow без \0!

3. Функции проверки класса символов <ctype.h>

1) int isalpha (int c)

Возвращает не NULL, если c-буква \Rightarrow

NULL - в противном случае.

2) int isupper (int c)

Возвращает не NULL, если c-буква верхнего регистра, NULL \Rightarrow в противном случае.

3) int islower (int c)

c-буква нижнего регистра

4) int isdigit (int c)

c-цифра \Rightarrow NULL

5) int isalnum (int c)

c-цифра/буква \Rightarrow NULL

6) int isspace (int c)

⌞, \t, \n, \f, \v, \r

c-пробел/пробелный символ \Rightarrow NULL

7) `int ispunct (int c)`

с-печатаемый символ, кроме пробела \Rightarrow `NULL`

8) `int isctrl (int c)`

(EOF)

с-управляющий символ \Rightarrow `NULL`

9) `int tolower (int c)`

с-буква верхнего регистра \rightarrow в нижний
регистр \Rightarrow ~~не~~ (не меняется)

10) `int toupper (int c)`

с-буква нижнего регистра \rightarrow в верхний
регистр \Rightarrow не меняется

4. Функции общего назначения `<stdlib.h>`.

1) `double atof (const char *s)`

Преобразует строку `s` в `double`

(Начальные символы до первого нецифрового, неч. пробела игнорируются).

2) `double atoi (const char *s)`

$s \in \text{int}$

3) `int atol (const char *s)`

$s \in \text{long int}$

4) `int rand (void)`
 (int) возвращает
число в диапазоне

gvanazore om 0 go max ^{в данной реализации} (не > 32767).

1) char *strcat (char s[], char t[])

{ int i=0, j=0;

while (s[i] != '\0')

i++;

while (s[i++] = t[j++] != '\0')

return s;

}

2) #include <ctype.h>

int atoi (char s[]);

{ int i, sign;

for (i=0; isspace(s[i]); i++)

;

sign = (s[i] == '-') ? -1 : 1;

if (s[i] == '+' || s[i] == '-')

i++;

for (n=0; isdigit(s[i]); i++)

n = 10 * n + (s[i] - '0');

return sign * n;

}

gama!

LL+48a62c

В C++ существует опред. возможность подкнчить некоторые конструкции для того, чтобы при возникновении искл. ситуации программа выдава сообщение или позволяла бы продолжить выполнение.

Способы обработки:

- Обработка исключения;
- Обработка завершения.

①. Обработка исключения

В программе можно установить контроль со стороны программиста над процессом преобразования строки.

В этом случае, если возникает искл. сит., то начинает работать другой кусок программы. Для этого надо выделить операторы в логический блок с помощью `{}`, а перед этим блоком `→ try`, а после проверяемого блока `→ catch`, введя за кторым (const название ситуации &)

`catch (...)` → как только возникла любая
искл. сит., управление в `catch`.

`try { операторы, вызывающие исключение }
catch (...){ операторы, выполняемые при
возникновении исключения }.`

Последовательность действий:

- 1) В блоке `try` заключаются операторы
в программе, которые, по мнению програм-
миста, могут вызвать исключение;
- 2) Если один из операторов вызывает
исключение, то аварийного завершения
не происходит, оставшиеся в блоке
операторы не выполняются;
- 3) Начинают выполнение операторы,
стоящие в блоке `catch`;
- 4) После завершения обработки искл. сит.,
работа программы продолжается с
оператора, стоящего сразу после
`try catch`.

5) Если все операторы в `try` выполнялись и исключения не возникло, то операторы `catch` - не выполняются.

II. Обработка завершения

`try { ... }`

`finally { ... }`

Отличие заключается в том, что операторы внутри блока `finally` будут выполняться всегда, независимо от того, возникло исключение в `try` или нет.

Последовательность:

- 1) Если в `try` возникло исключение, то после выполнения операторов в `finally` программа аварийно завершается;
- 2) Если исключения не возникло, программа продолжит работу после `finally`;
- 3) Данная конструкция не подавляет исключений, поэтому, если требуется

продолжить выполнение программы
после возникновения исключения, необхо-
димо поместить конструкцию в
try внешней конструкции try catch.

Преднамеренная генерация
исключений.

В программах, где обработка результатов
выполнения тех или иных операций
ведется с применением механизмов
обработки исключений, существует возмож-
ность искусственной генерации исключ. ситу-
аций с целью уведомления программы
о неудачном выполнении заданных действий.
Генерирует исключения \rightarrow throw, за которыми
следует необяз. выражение

В программе организован цикл, который
должен выполняться 3 раза. В нём \rightarrow
блок, генерирующий исключение различных
типов (char, int, double) в зависимости от
i.

На первом проходе `throw` выбрасывает `min char`, перехватываемый вторым обработчиком. На 2-ом проходе функция `→ throw int`, перехватываемая первым обработчиком. На 3-ем этапе функция `→ throw double`, для которой обработчик не предусмотрено. Но имеется «всегдашний» 3-ий обработчик.

1.04

функция

7.

②.6 Особенности описания переменных

В C++ переменную можно определить в любом месте программы, а не только в начале функции/блока. Это полезно тем, что переменную можно описать в непосредственной близости от места её

вание типов не является синтаксической ошибкой.

Объединение может являться компонентом структурной переменной, и, наоборот, одним из компонентов объединения может быть структурная переменная.

Обработка искл. ситуаций.

29.03

При написании программы могут встретиться синтаксические ошибки, которые обнаруживает компилятор. Но бывают ошибки, которые не предусмотрены пользователем. От этих ситуаций можно избавляться, делая множество проверок, что делает программу менее понятной и более громоздкой. Если таких проверок нет, то при подобных ошибках возникает исключ. ситуация, при которой выполнение дальнейших вычислений прекращается.

Семинар
4