Семинар 5. Стадии препроцессинга, замены в тексте, включение текстов из файлов, условная компиляция, макроподстановки, препроцессорные операции и дополнительные директивы, встроенные макроимена

1. Стадии препроцессинга

Препроцессор – обязательный компонент компилятора.
Назначение препроцессора – обработка исходного текста программы до её компиляции. Результат препроцессорной обработки – единица трансляции.

1. Стадии препроцессинга

Стадии:

- системно-зависимые обозначения (например, индикатор конца строки) перекодируются в стандартные коды
- пара символов «\» и «конец строки» убираются, следующая строка присоединяется к строке, где находилась эта пара символов.

1. Стадии препроцессинга

Пример.

```
int main()
{
    cout << "hello" \
    "world";
    int a = 10,\
    b = 11;
    cout << endl << "a+b=" << a+b;
    getchar();
    return 0;
}
```

1. Стадии препроцессинга

Стадии (продолжение):

- в тексте распознаются директивы препроцессора, а каждый комментарий заменяется одним символом пробела
- выполняются директивы препроцессора и производятся макроподстановки
- ескейп-последовательности (напр., '\n') заменяются на их числовые коды

1. Стадии препроцессинга

Стадии (продолжение):

• смежные строковые константы конкатенируются (т.е. заменяются на одну строковую константу)

Пример. Объединение смежных строковых констант:

```
int main()
{
    cout << "Today"  " is " "Tuesday";
    getchar();
    return 0;
}
```

1. Стадии препроцессинга

На стадии обработки директив препроцессора:

- замена препроцессорных констант
- включение в программу текстов из указанных файлов
- исключение из программы отдельных частей её текста (условная компиляция)
- макроподстановка (замена обозначения параметзированным текстом)

1. Стадии препроцессинга

Для управления препроцессором используются команды (директивы) препроцессора:
#define, #include, #undef, #if, #ifdef, #ifndef, #else, #endif, #line, #error, #pragma, #
Каждая препроцессорная директива завершается переходом на новую строку.

1. Стадии препроцессинга

Директива	Действие
#define	Определение препроцессорного идентификатора или макроса
#undef	Отменяет действие команды #define
#include	Включает в текст программы текст из выбранного файла
#if, #ifdef, #ifndef, #else, #endif, #elif	Организуют условную обработку текста программы
#line	Управляет нумерацией строк
#error	Задаёт текст диагностического сообщения об ошибке
#pragma	Вызывает действия, зависящие от реализации
#	Ничего не вызывает, т.к. является пустой

2. Замены в тексте

Синтаксис:

#define идентификатор строка_замещения

Пример:

Исходный текст	Результат препроцессорной обработки

#define K 40
int main(){
  int M[K][K];
  double A[K];
	
	int main(){
	  int M[40][40];
	  double A[40];

2. Замены в тексте

Препроцессорные замены, предусмотренные директивой #define, не выполняются внутри строковых и символьных констант и комментариев, т.е. не распространяются на тексты, ограниченные кавычками ("), апострофами (') и разделителями (/*, */). В то же время строка замещения может содержать эти ограничители.

2. Замены в тексте

Замены в тексте можно отменять с помощью команды:

#undef идентификатор

После этой директивы идентификатор для препроцессора становится неопределённым, и его можно определять повторно с помощью другой команды #define.

2. Замены в тексте

Пример.

```
#include <iostream>
using namespace std;
int main()
{
    #define K 50
    #define MESSAGE cout << "K=" << K << endl;
    MESSAGE
    #undef K
    #define K 30
    MESSAGE
    getchar();
    return 0;
}
```

2. Замены в тексте

Директиву #undef удобно использовать при разработке больших программ из «кусков кода», написанных разными программистами. В этом случае могут встретиться одинаковые идентификаторы для разных объектов. Чтобы не изменять исходных файлов, включаемый текст можно обрамлять директивами #define - #undef.

2. Замены в тексте

Пример.

```
#include <iostream>
using namespace std;

int main()
{
	int A = 10;
	#define A X // Начало включения
	int A; // Включенный текст
	A = 5; // Включенный текст
	cout << "A=" << A << endl; // Включенный текст
	#undef A // Конец включения
	cout << "A=" << A << endl;
	getchar();
	return 0;
}
```

3. Включение текстов из файлов

Для включения текстов из файлов используется препроцессорная директива #include, имеющая две формы записи:

```
// Для поиска в стандартных системных каталогах
#include <имя_файла>
// Для поиска сначала в текущем каталоге:
// (только затем просматривается системный каталог)
#include "имя_файла"
```

3. Включение текстов из файлов

Пример использования внешних переменных:

Header1.h	Proj.cpp

int externVariable1 = 10;
double externVariable2 = 0.5;
	#include "stdafx.h"
	#include <iostream>
	using namespace std;
	#include "Header1.h"
	extern int externVariable1;
	extern double externVariable2;
	void main()
	{
	  cout << "Extern variables:" << endl;
	  cout << externVariable1 << ' ' << externVariable2;
	  getchar();
	}

4. Условная компиляция

Команды условной компиляции:

#if константное_выражение

#ifdef идентификатор

#ifndef идентификатор

#else

#endif

#elif

4. Условная компиляция

Общая структура применения директив условной компиляции:

```
#if константное_выражение
Текст_1
#else
Текст_2
#endif
```

Текст_1 включается в единицу трансляции только если константное выражение истинно. Текст_2 – только если константное выражение ложно. Если директива #else отсутствует и константное выражение ложно, то весь текст от #if до #endif не входит в единицу трансляции (не будет откомпилирован).

4. Условная компиляция

Вместо директивы #if могут быть использованы директивы #ifdef, #ifndef.
В директиве #ifdef проверяется, определён ли с помощью команды #define к текущему моменту идентификатор, помещённый после #ifdef. В директиве #ifndef проверяется обратное условие (истинным считается неопределённость идентификатора).

4. Условная компиляция

Условную компиляцию удобно применять при отладке программы для вывода информации:

```
#define OTLADKA 1
…
#ifdef OTLADKA
cout << "Отладочная печать";
#endif
```

4. Условная компиляция

Также условную компиляцию применяют для защиты текста header-файлов от повторного включения:

```
#ifndef _FILE_NAME
// некоторый код
#define _FILE_NAME 1
#endif
```

_FILE_NAME – зарезервированный программистом для файла препроцессорный идентификатор

4. Условная компиляция

Для организации мультиветвления во время обработки препроцессором кода программы применяется директива
#elif константное выражение

5. Макроподстановки

Макрос – это средство замены одной последовательности символов другой последовательностью.

Простейшее макроопределение:
#define идентификатор строка замещения

Возможно макроопределение с параметрами:
#define имя(список_параметров) стр_зам

5. Макроподстановки

Пример макроопределения с параметром:

```
#define max(a,b) (a<b?b:a)
```

Вызов макроопределения:

```
cout << max(5,-10);
// max(5,-10) заменяется на (5<-10?-10:5)
```

5. Макроподстановки

Ещё один пример макроопределения с параметром:

```
#define ABS(x) (x<0?-(x):x)

void main()
{
cout << "ABS(-10)=" << ABS(-10) << endl;
cout << "ABS(10)=" << ABS(10) << endl;
getchar();
}
```

5. Макроподстановки

Отличия макросов от функций:

- коды вставляются в программу столько раз, сколько раз используется макрос
- подстановка для макроса используется всегда (отличие от inline-функций)
- макрос пригоден для обработки аргументов любого типа, допустимых в выражениях, формируемых при обработке строки замещения

5. Макроподстановки

Будьте осторожны с макросами:

```
#define max(a,b) (a<b?b:a)
#define t(e) e*3
#define PRINT(c) {cout << #c << " is equal " << c << endl;}
void main()
{
    int x = 2;
    PRINT(x);
    PRINT(max(++x, ++x));
    PRINT(t(x));
    PRINT(t(x+x));
    PRINT(t(x+x)/3);
    getchar();
}
```

5. Макроподстановки

Будьте осторожны с макросами:

```
#define max(a,b) (a<b?b:a)
#define t(e) e*3
#define PRINT(c) {cout << #c << " is equal " << c << endl;}
```

```
x is equal 2
max(++x, ++x) is equal 5
t(x) is equal 15
t(x+x) is equal 20
t(x+x)/3 is equal 10
```

5. Макроподстановки

Будьте осторожны с макросами:

Вызов макроса	Результат подстановки
max(++x, ++x)	(++x < ++x ? ++x: ++x)
t(x)	x*3
t(x+x)	x+x*3
t(x+x)/3	x+x*3/3

Для утроения аргумента вместо

```
#define t(e) e*3
```

нужно использовать

```
#define t(e) (e)*3
```

5. Макроподстановки

В случае необходимости подставляемое значение аргумента макроса можно заключить в строке замещения в кавычки (""). Для этого используется специальная операция #, записываемая непосредственно перед параметром. Пример:

```
#define PRINT(c) {cout << #c << " is equal " << c << endl;}
```

PRINT(x) будет заменено на:

```
cout << "x" << " is equal " << x << endl;
```

5. Макроподстановки

Пример. Макрос для печати массива.

```
#define ARR_PRINT(ARRAY, N) \
{    int i; \
    for(i=0; i<sizeof(ARRAY)/sizeof(ARRAY[0]); i++) \
    {\
        cout << #ARRAY"[" << i << "]=" << ARRAY[i] << '\t'; \
        if((i+1) % N == 0) cout << endl; \
    } \
}
```

5. Макроподстановки

Пример. Макрос для печати массива.

```
int arr[] = {8, 5, 4, 45, 55, 75, 5476, 6, 64, 43};
ARR_PRINT(arr,3);
```

6. Препроцессорные операции и дополнительные директивы

Операция	Действие операции
#	текст, замещающий данный параметр, заключается в кавычки
##	конкатенация лексем строки замещения
defined	позволяет заменить #ifdef и #ifndef на #if defined и #if !defined (обратите внимание: в этом случае скобки после #if должны отсутствовать)

Пример для операции ##:

```
#define abc(a,b,c,d) a##(##b##c##d)
```

`abc(sin,x,'+',y)` будет заменено на `sin(x+y)`

6. Препроцессорные операции и дополнительные директивы

Директива

#line целочисленная_константа

указывает компилятору, что следующая строка имеет номер, заданный приведённой константой.

Возможно задание как номера строки, так и имени файла:

#line константа "имя_файла"

6. Препроцессорные операции и дополнительные директивы

Директива

#error последовательность лексем

в случае выполнения приводит к неудачной компиляции и выдаче диагностического сообщения. Пример:

```
#define NAME 4
...
#if (NAME != 4)
#error NAME must be equal to 4!
#endif
```

7. Встроенные макроимена

Макроимя	Информация
__LINE__	Номер текущей строки
__FILE__	Имя компилируемого файла
__DATE__	Строка в формате "Mmm dd yyyy", определяющая дату начала обработки исходного файла
__TIME__	Строка в формате "hh:mm:ss", определяющая время начала трансляции текущего исходного файла
__STDC__	Константа, равная 1, если компилятор работает в соответствии с ANSI-стандартом.

7. Встроенные макроимена

Пример. Использование макроимён.

```
void main()
{
cout << "__DATE__ = " << __DATE__ << endl;
cout << "__TIME__ = " << __TIME__ << endl;
#line 1024 "example.cpp"
cout << "__LINE__ = " << __LINE__ << endl;
cout << "__FILE__ = " << __FILE__ << endl;
getchar();
}
```

8. Задания

1. Создайте макрос для возведения числа в любую целую степень без использования функций math.h (подсказка: используйте цикл for). Итог должен выводиться на экран.
2. Создайте макрос для печати всех чётных чисел в определённом заданном промежутке.
3. Создайте макрос для подсчёта количества чётных и нечётных чисел в целочисленном одномерном массиве.