**ПРЕПРОЦЕССОР ЯЗЫКА С**

В интегрированную среду подготовки программ на С++ или компилятор языка как обязательный компонент входит **ПРЕПРОЦЕССОР**.

**Препроцессор** используется для обработки текста программы до ее компиляции и вызывается автоматически при обращении к компилятору. Обработка заключается в выполнении специальных команд (Директив препроцессора).

Во время стадии обработки выполняются следующие действия:

-Замена идентификаторов (обозначений) заранее подготовленными последовательностями.

- Включение в программу текстов из указанных файлов.

- Исключение из текста программы отдельных частей ее текста (условная компиляция).

- Макроподстановка, то есть, замена обозначения параметризованным текстом, формируемым препроцессором с учетом конкретных параметров.

Для управления процессором, то есть, для задания нужных действий, используются команды (директивы) препроцессора, каждая из которых помещается в отдельной строке и помечается символом ‘#’.

 Рассмотрим наиболее часто используемые команды препроцессора.

**6.1 Команда #include**

Команда #include используется для включения в исходный текст программы файлов, содержащих прототипы стандартных функций и части исходных текстов программы. Данное средство языка позволяет хранить в отдельных файлах описания констант, типов и функций и затем использовать их в разных программах.

Команда встречается в двух вариантах:

**#include <Имя файла>** – в этом случае файл ищется в каталогах среды – эта форма используется при подключении стандартных файлов Си и С++;

**#include ″Имя файла″** – в этом случае файл сначала ищется в текущем каталоге проекта, а потом уже в каталогах среды – такая форма используется при подключении файлов конкретного проекта.

**6.2 Команды #define и #undef**

Команда #define используется для описания макроопределений – текстовых строк, вставляемых препроцессором вместо указанных имен (макрокоманд). Вставляемые фрагменты можно настраивать с использованием специально задаваемых параметров. Компиляция строк, как уже упоминалось ранее, выполняется после настройки и подстановки фрагментов.

Команда записывается в следующем виде:

**#define <Идентификатор>[(<Список параметров>)] <Строка>**

где **<Идентификатор>** – имя, используемое для идентификации строки (макрокоманда);

 **<Список параметров>** – список идентификаторов, обозначающих заменяемые элементы строки – может отсутствовать;

 **<Строка>** – строка-макроопределение, заменяющая имя в тексте программы.

Примеры:

а) #define nil 0 // определяет замену идентификатора nil нулем

б) #define N 5 // определяет замену идентификатора N числом 5

 int main()

 {

 int A[N][N]; ⇒ A[5][5]

 …

в) **#define MSG ″Это строка - пример″** // определяет замену

 // идентификатора MSG заданной строкой

г) **#define P(X) printf(″X равен %d.\n″,X)** // определяет замену

 // идентификатора с параметром X на строку вывода данного параметра,

 // например, вместо **P(alf)** будет вставлено **printf(″alf равно %d.\n″,alf)**

Следует осторожнее использовать форму с параметрами, так как в отличии от inline функций, препроцессор осуществляет формальную подстановку параметров как символьных строк без проверки типов. Например, рассмотрим два варианта определения функции поиска максимального значения:

а) **inline int MAX(int X, int Y){return X>Y?X:Y;}**

б) **#define MAX(X,Y) (X>Y?X:Y)**

Вариант а не способен работать с типами, отличными от int, в то время как вариант б может принимать аргументы любых типов.

***Примечание.*** Следует заметить, что подобное использование Си-команды #define для C++ не рекомендуется, поскольку С++ имеет другие механизмы, например шаблоны (template), позволяющие выполнять контроль типов:

**template <typename T> T MAX(T x, T y){ return x>y?x:y;}**

Вызов функции при использовании шаблона можно осуществлять либо с неявным указанием типа, например MAX(1,4), либо если аргументы имеют разный тип, с явным указанием типа MAX<float>(1.03, 4).

Формальные подстановки, выполняемые посредством #define, могут привести к неожиданным ошибкам, например:

#define ABS(X) (X>0?X:-X)

void main()

{

 int i,j; … i= ABS(j++); // Ошибка! j++ будет выполнено дважды

Если подстрока в команде #define не указана, то она получает значение «пусто» и при обработке описанное таким образом имя макроса из текста удаляется. Однако соответствующая константа считается определенной и при проверке командой **if defined** (или **ifdef**) или **if !defined** (или **ifndef**) возвращает «Константа определена».

Команда **#undef** используется для отмены команды **#define.** Формат команды:

**#undef <Имя>**

После выполнения этой команды замена идентификатора на строку прекращается.

**6.3 Команды условной компиляции**

Специальные команды препроцессора позволяют осуществлять условную генерацию текста программы. При этом используются две конструкции:

**#if <Константное выражение>**

<Операторы языка и команды препроцессора>

**[#else**

<Операторы языка и команды препроцессора>]

**#endif**

или

**#if <Константное выражение>**

<Операторы языка и команды препроцессора >

**#elif <Константное выражение>**

<Операторы языка и команды препроцессора>

**[#else**

<Операторы языка и команды препроцессора>]

**#endif**

При этом в обоих случаях сначала осуществляется проверка заданного константного выражения. Если результат этого выражения отличен от нуля, то в текст вставляются следующие за **#if** операторы языка и команды препроцессора. Если результат выражения равен 0, то в первом варианте в текст вставляются операторы языка и команды препроцессора, следующие за **#else,** а во втором варианте – осуществляется дополнительная проверка второго константного выражения. И если результат проверки второго выражения отличен от нуля, то в текст программы вставляются операторы языка и команды препроцессора, следующие за **#elif**, иначе в текст вставляются операторы языка и команды препроцессора, следующие за **#else.** Альтернатива **#else** в обоих вариантах может быть опущена.

Таким образом, команды условной генерации позволяют изменять исходный текст программы в зависимости от значений некоторых констант.

Команда **#error** используется для вывода сообщений об особенностях генерации.

Формат команды

 **#error <Сообщение>**

Команды **#ifdef и #ifndef** используются как альтернативы соответственно:

**#if defined(<Идентификатор>)** и **#if !defined(<Идентификатор>)**, где

**defined** – специальная функция препроцессора, возвращающая 1, если заданный идентификатор определен командой #define и это определение не отменено командой

**#undef.**

С использованием этих команд выполняется, например, защита от повторной компиляции подключаемых файлов:

#ifndef MyTerm\_h // если не определена константа

#define MyTerm\_h // определяем константу как пустую

… Подключение файлов и исходные тексты …

#endif

Первый раз константа не определена, и текст внутри фрагмента подключается. Однако уже при первом входе во фрагмент константа определяется посредством #define. Соответственно следующий раз константа уже будет определена, и фрагмент подключаться не будет.

Ниже приведены еще два примера использования условной макрогенерации:

а) **#if !define(MODEL)**

 #error Building model not defined

 **#endif**

б) **#if defined(NEARPOINTERS)**

 space=farcoreleft();

 **#elif defined(FARPOINTERS)**

 space=coreleft();

 **#else**

 **#error Unsupported memory model**

 **#endif**

**6.4 Некоторые предопределенные макроопределения**

**DATE** – текстовая строка, содержащая дату компиляции в формате Mmm dd yyyy, где

Mmm – название месяца, dd – число и yyyy – год.

**FILE** – текстовая строка, содержащая имя компилируемого файла, включая полный путь к нему.

**LINE** – номер текущей строки в десятичном формате. Может быть изменен директивой **#line**.

**TIME** – текстовая строка, содержащая время компиляции в формате **hh:mm:ss**, где hh – часы, mm – минуты, ss - секунды.

**TIMESTAMP** – текстовая строка, содержащая дату и время последнего изменения

компилируемого файла, в формате Ddd Mmm Date hh:mm:ss yyyy, где Ddd – сокращение названия дня недели, Date – число.

**FUNCTION** – текстовая строка, содержащая название функции, внутри которой

вставлено макроопределение.

**Пример.** Создание макрокоманды отладочной печати:

**#ifdef \_DEBUG**

 **#define DEBUG\_OUTPUT \**

 **printf( "%s\t%s\t%d\n",\_\_FILE\_\_,\_\_FUNCTION\_\_,\_\_LINE\_\_ );**

**#else**

 **#define DEBUG\_OUTPUT**

**#endif**

При наличии такой макрокоманды в тексте программы, где необходим отладочный вывод, достаточно вставить строку вида:

**DEBUG\_OUTPUT("file error…");** чтобы было сгенерировано детальное сообщение об ошибке.

**STDC** – наличие макроопределения сообщает о том, что компилятор удовлетворяет

требованию стандарта ANSI C.

**cplusplus** – определено в том случае, если компилируется программа на C++.

**Пример.** Подключение заголовочного файла **bdb.h**, написанного на языке Cи,

независимо от того, компилируется ли Cи или C++ программа:

**#ifdef \_\_cplusplus**

**extern "C"{**

**#endif**

**#include <bdb.h>**

**#ifdef \_\_cplusplus**

**};**

**#endif**

***Примечание.*** Необходимость отдельно обрабатывать Cи-заголовочные файлы связана с тем, что в C и C++ используются различные схемы преобразования имен функций при компиляции. Поэтому библиотека, полученная компилятором Cи, может быть подключена к С++ программе только в том случае, если имена функций совпадают, что и обеспечивает директива "C".

**DEBUG** – определено в том случае, если компилируется отладочная версия

программы.

**MSC\_VER** – содержит версию компилятора. Microsoft Visual C++ .NET 2003 – 1310,

где 13 – номер версии, 1.0 - реализация. Visual C++ 2005 – 1400, Visual C++ 2008 - 1500.

**WIN32** – определено при компиляции программ для Win32 и Win64.

**WIN64** – определено при компиляции программ для Win64.