Условная компиляция

C / C++

Условная компиляция выполняется с помощью специальных директив и позволяет выборочно компилировать части исходного кода программы.

В C/C++ эти директивы входят в состав препроцессора, назначение которого – обработка исходного текста программы до ее компиляции (на первой фазе компиляции).

Директивы препроцессора размещается в любом месте программы (каждая в отдельной строке).

Они начинаются с символа #, перед которым в строке могут быть только пробелы, например, #define MAX 10.

Обработки директив препроцессора обычно включает следующие действия:

- замена идентификаторов заранее подготовленными последователь-
- ностями символов;
- включение в программу текстов из указанных файлов;
- исключение из программы отдельных частей ее текста, условная компиляция;
- макроподстановка, то есть замена обозначения параметризованным текстом, формируемым препроцессором с учетом конкретных аргументов.

Директивы условной компиляции

C / C++

Директивы #if, #else, #elif, #endif – позволяют в зависимости от константного выражения включать или исключать те или иные части кода программы. **Формат:**

```
#if <константное выражение>
  <операторы>…
     [#elif <константное выражение 1>
       <операторы>…]
     [#elif <константное выражение 2>
  <операторы>…]
  <операторы>…
 [#else
    <операторы>…]
#endif
```

Если находящееся за if константное выражение истинно, то компилируется код расположенный между #if и #endif, иначе компилируется код после блока #else код.

#elif – это else if – оператор организующий цепочку if-else-if, если константное выражение 1 – истинно, то компилируются операторы этого субблока и происходит переход к else. Иначе проверяется выражение 2 и т.д.

Вложенность директив #if и #elif в языке C – до 8-и уровней, а в C++ – до 256.

В константных выражениях можно проверять, например, определена ли константа:

```
#if defined(_BORLANDC_) && _BORLANDC_ == 0x530
```

Директивы условной компиляции

C / C++

Пример: использование

```
#if
  #if MAX > 100
                              #if
SERIAL_VERSION
          int port = 198;
      #elif
          int port = 200;
  #else
      char out_buffer[100];
  #endif
```

Директивы #ifdef, #ifndef – позволяют управлять компиляцией в зависимости от того, определен ли с помощью директивы #define указанный в них идентификатор.

#ifdef – if define (если определено), #ifndef – if not define (если не определено). Формат:

```
#ifdef <имя_макроса>
 <операторы>… //код компилируется, если
#endif          // <имя_макроса>   определено
#ifndef <имя_макроса>
 <операторы>… // код компилируется, если
#endif          // <имя_макроса> не определено
```

Вложенность директив #if и #elif в языке C – до 8-и уровней, а в C++ – до 256.

Часто эти директивы используют для объявления заголовочного файла только один раз (обычно в заголовочных файлах используется макросы и проверяют их): #ifndef CMPL

```
#include "cmplx.h"
#define CMPL
#endif
```

Макрокоманды (макросы)

C / C++

Макрокоманды (макросы) являются расширением языка и имеются во всех развитых языках программирования, как в высокоуровневых (Си/Паскаль), так и в Ассемблере. Макрокоманда служит для представления часто встречающегося однотипного кода под одним именем. В отличие от функции, которая существует в одном экземпляре и просто вызывается когда это необходимо, содержимое макрокоманды подставляется вместо её имени всякий раз когда она встречается. Способом применения макрокоманда похожа на константу, где имя константы заменяется соответствующим ей значением при компиляции.

В C/C++ макрокоманды объявляется в директиве препроцессора

#define <имя_макроса> [(<параметры>)] <текст_подстановки>

В теле программы используется <имя_макроса>, которое при препроцессорной обработке заменяется на <текст_подстановки>, который может быть константой или значением выражения.

Примеры:

```
#define true 1
#define false 0
```

Определение констант

```
#define CUBE(x) ((x)* (x)* (x))
…….
result = CUBE(g);
```

Подстановка макрокоманды

- Макрокоманды и именованные константы облегчают чтение ваших программ, заменяя сложные выражения и числовые константы смысловыми именами. Они позволяют расширить синтаксис языка новыми ключевыми словами.
- В процессе компиляции компилятор C/C++ использует препроцессор, для замены каждой именованной константы или макрокоманды соответствующим значением (выполняет макроподстановку – macro replacement).
- Макрокоманды выполняются быстрее функций, но увеличивают размер выполняемой программы.
- Если определение макрокоманды выходит за пределы одной строки, надо в конце каждой строки поместить символ обратного слэша - \ , чтобы информировать препроцессор о том, что определение продолжается на следующей строке.

Макрокоманды (макросы)

C / C++

Предопределенные макросы

В языке C/C++ существует шесть встроенных (стандартных) имен макросов:

_ _FILE_ _ - строка с полным именем текущего файла

_ _LINE_ _ - номер текущей строки компилируемого кода

Содержимое этих макросов изменяется директивой #line <номер> [<имя_файла>],
где <номер> - положительное число, задающее номер текущей строки,
а <имя_файла> - любое допустимое имя файла, задающее имя текущего файла.

_ _DATE_ _ - дата компиляции файла, в формате месяц/день/год

_ _TIME_ _ - время компиляции файла, в формате часы:минуты:секунды

_ _STDC_ _ - макрос определен, когда компилятор работает в режиме совместимости с ANSI C-компилятором, иначе – макрос не определен (STandarD C).

_ _cplusplus - макрос определен, если используется компилятор C++, если используется компилятор C – макрос не определен.