Учебный Центр компьютерного обучения «СПЕЦИАЛИСТ» при МГТУ им. Н.Э.Баумана

Язык программирования C

В.Г.Тетерин – Microsoft Solution Developer (Visual C++)
teterin@specialist.ru

Центр компьютерного обучения «СПЕЦИАЛИСТ» при МГТУ им. Н.Э.Баумана®

МОДУЛЬ 5

ПРЕПРОЦЕССОР

Модуль 5. Препроцессор

- Препроцессорные директивы

  `#include`

  `#define`

  `#undef`

  `#if - #else - #endif`

- Макроопределения с параметрами

- Правила оформления деклараций

Директивы препроцессора

- Прежде, чем поступить на вход компилятора, исходный текст программы на языке С обрабатывается специальной утилитой - *препроцессором*.
- Работа препроцессора управляется специальными *директивами* - инструкциями, которые включаются в исходный текст программы на языке С.
- Эти директивы не являются частью языка программирования.
- Каждая директива начинается с символа **#** и заканчивается символом новой строки.
- Точка с запятой (;) в конце директивы не применяется.

Директива #include

- Директива **#include** - включение файла
  - Форматы директивы:
    - **#include "имя_файла"**
    - **#include <имя_файла>**
  - Вместо директивы **#include** в текст программы включается содержимое указанного файла.
  - При использовании первой формы оператора поиск файла осуществляется в текущем каталоге, а затем, если он там не обнаружен - в других "стандартных" каталогах.
    - Пример:

```
#include "student.h"
```

  - При второй форме оператора поиск файла производится сразу в "стандартных" каталогах, а текущий каталог просматриваться не будет.
    - Пример:

```
#include <stdio.h>
```

  - Включаемые файлы в свою очередь могут содержать директивы **#include**.

Директива #define

- Директива **#define** – макроопределение (макроподстановка)

  - Форматы директивы:

```
#define идентификатор подстановка

#define идентификатор(формальные_аргументы) подстановка
```

  – Директива предписывает всюду в тексте программы, начиная с текущей точки и до конца файла или до директивы **#undef**, выполнять замену указанного идентификатора на сопоставленную ему цепочку символов (подстановку).

    - Часто процесс подстановки называется *макрорасширением*.

  – При второй форме директивы во время макроподстановки формальные аргументы, входящие в цепочку-подстановку, заменяются их фактическими значениями.

    - В определении такой параметрической макроподстановки между идентификатором и открывающей круглой скобкой не должно быть пробелов.

Директива #define (продолжение)

- Примеры:

```
#define MAXLEN 128

#define MAX(x,y) ( (x)>(y) ? (x) : (y) )
```

- Длинная цепочка-подстановка может быть продолжена на следующей строке, для этого в конце строки, имеющей продолжение, ставится символ \.

- После выполнения текущей макроподстановки получившаяся строка вновь просматривается на предмет возможных макроподстановок.

- Макроподстановка, заданная директивой **`#define`**, не будет выполняться, если встретившийся идентификатор является частью *символической константы* или строки символов, заключенной в *двойные кавычки* или *угловые скобки*.

- В С принято записывать идентификатор, определяемый директивой **`#define`**, *прописными буквами*, это является напоминанием, что будет выполняться макроподстановка.

Директива #define (продолжение)

- Язык С содержит *предопределенные макроимена*, которые не допускается использовать в директиве #define, их обозначения начинаются и заканчиваются двумя символами подчеркивания:

__STDC__ - имеет значение 1, если компиляция выполняется в режиме совместимости с ANSI-стандартом, в противном случае является неопределенным;

__FILE__ - литерал - имя текущего обрабатываемого компилятором файла;

__LINE__ - номер текущей обрабатываемой компилятором строки файла;

__DATE__ - дата начала обработки текущего файла - литерал в формате mmm dd yyyy;

__TIME__ - время начала обработки текущего файла в формате hh:mm:ss.

Директива #define (продолжение)

- В цепочке-подстановке можно использовать символы ## с необязательным пробелом с каждой стороны, это приводит к "склеиванию" элементов при макроподстановке.
  - Такое свойство применяется для конструирования имен, например, макроопределение

```
#define VAR(i,j) i##j
```

    вызовет в тексте программы замену выражения `VAR(x,6)` на `x6`.

- Одинарный символ # перед макроаргументом в цепочке-подстановке вызывает преобразование фактического аргумента в символьную строку
  - например, макроопределение

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

    приведет в тексте программы

```
val = 1024;  TRACE(val);
```

    к такой последовательности замен: `TRACE(val)` превратится вначале в

```
printf("val" "=%d\n",val);
```

    и затем, после конкатенации двух строк, окончательно получится

```
printf("val=%d\n",val);
```

Директива #undef

- Директива **#undef** - отмена макроопределения
  - Формат директивы:

    #undef идентификатор
  - Эта директива отменяет определение указанного идентификатора, он будет считаться неопределенным и более не подлежащим замене.
  - Примеры:

```
#undef MAXLEN

#undef MAX
```

Директивы условной компиляции

- Формат директив:
  - `#if константное_выражение`
  - `#ifdef идентификатор`
  - `#ifndef идентификатор`
  - `#elif константное_выражение`
  - `#else`
  - `#endif`
- Эти директивы обеспечивают компиляцию тех или иных участков программного кода в зависимости от выполнения некоторых условий.
- Для проверки условия применяется одна из первых трех директив, проверяющих истинно ли (`==1`) константное_выражение (`#if`), определен (`#ifdef`) или неопределен (`#ifndef`) указанный идентификатор.
- Если условие выполнено, то компилируются операторы программы, расположенные вслед за проверяющей директивой вплоть до встречи одной из оставшихся трех директив.
- Если условие не выполнено, то эта часть программы игнорируется.

Директивы условной компиляции

- Формат директив:
  - `#if константное_выражение`
  - `#ifdef идентификатор`
  - `#ifndef идентификатор`
  - `#elif константное_выражение`
  - `#else`
  - `#endif`
- Необязательная директива **`#else`** вводит альтернативную ветвь программы, которая будет компилироваться, если не выполнено условие в непосредственно предшествующей ей проверяющей директиве.
- Необязательная директива **`#elif`** эквивалентна последовательности следующих одна за другой директив **`#else`** и **`#if`** и применяется для организации множественного ветвления.
- Директива **`#endif`** отмечает конец участка условной компиляции, начатого одной из первых трех директив проверки.
- Константное выражение может состоять только из тех объектов, которые к моменту проверки уже определены.

Директивы условной компиляции (продолжение)

- Язык С позволяет использовать в константном выражении специальную операцию

```
defined(идентификатор)
```

которая принимает значение 1 (истина), если идентификатор определен, и 0 (ложь) - в противном случае.

Таким образом, директива

```
#if defined(symbol)
```

эквивалентна

```
#ifdef symbol
```

Операция defined позволяет организовать сложные проверки, например:

```
#if defined(symbol1) || defined(symbol2)
```

- Конструкции, образованные директивами условной компиляции могут вкладываться одна в другую.

Директивы условной компиляции (продолжение)

- Основное применение директив условной компиляции – разработка переносимого между платформами кода, главным образом – библиотек.
- Директивы условной компиляции также применяются для защиты заголовочных файлов (хедер-файлов, .h-файлов) от повторного включения (от повторной загрузки)
- Пример:
  - Для защиты файла **`student.h`** в него следует включить директивы

```
#ifndef STUDENT_H
#define STUDENT_H
// содержимое файла
#endif
```

  - Вместо директивы

```
#ifndef STUDENT_H
```

    можно использовать

```
#if !defined(STUDENT_H)
```

Прочие директивы

- Директива **#line**
  - Форматы директивы:

```
#line номер_строки

#line номер_строки имя_файла
```

  - Эта директива предписывает заменить значения предопределенных макроимен **__LINE__** и **__FILE__** на указанные значения.
- Директива **#error**
  - Формат директивы:

```
#error сообщение
```

    Директива предписывает прекратить процесс компиляции с выдачей сообщения следующего вида:

```
Fatal: имя_файла номер_строки Error directive: сообщение
```

  - Используется для целей отладки, например,

```
#if (VAL != 0 && VAL != 1)

#error VAL must be defined to either 0 or 1

#endif
```

Прочие директивы (продолжение)

- Директива **#pragma**
  - Формат директивы:

    #pragma инструкция

  - Назначение этой директивы - передача специфических, зависящих от реализации, инструкций и обеспечение переносимости программы в другие системы программирования на языке С
    - по определению, всякий компилятор языка С, поддерживающий директиву **#pragma**, будет ее игнорировать, если он не может распознать содержащуюся в ней инструкцию.

Итоги

- В этом модуле Вы изучили:
  - Директивы препроцессора
  - Макроопределения `#define` с параметрами
  - Применение директив условной компиляции для защиты заголовочных файлов от повторной загрузки

Вопросы?

- В.Г.Тетерин – Microsoft Solution Developer (Visual C++)
- teterin@specialist.ru

ПРИЛОЖЕНИЕ

ЗАДАЧИ

Задачи

1. Написать программу «Калькулятор»:
   1. примитивный:
      - вводится формула вида 12.5 + 32.56.
      - напечатать ответ (операции: +, -,*,/).
   2. кнопочный:
      - вводится произвольная формула вида 2.5 + 32 *5.1 =
      - напечатать ответ (без учета приоритетов операций).
        – Усложненный вариант – печатать при вводе формулы промежуточные результаты вычислений (например, в квадратных скобках) после ввода каждого знака операции (может быть, кроме первого)
   3. анализатор приоритетов:
      - вводится произвольная формула вида 2.5 + 32 *5.1 =
      - напечатать ответ с учетом приоритетов операций (формула не содержит скобок)

Задачи

4. анализатор формул
   - вводится произвольная формула вида 3*(7+(2.5 + 32) *(5.1-2))=
   - напечатать ответ с учетом приоритетов операций и вложенности скобок

2. Напечатать только те числа из входного потока, которые не содержат одинаковых цифр:
   2. числа натуральные.
   3. числа целые (могут быть отрицательными).
3. Найти первые 20 троек пифагоровых чисел ($a^2 + b^2 = c^2$).
4. Найти первые N натуральных чисел – палиндромов.
5. Найти первые N натуральных чисел, которые делятся на каждую свою цифру.

Задачи

6. Найти первые N натуральных чисел, цифры в записи которых образуют строго возрастающую последовательность.
7. Среди первых N натуральных чисел напечатать те из них, которые являются совершенными, т.е. равны сумме всех своих простых делителей, включая 1, например, 6=1+2+3.
8. Шестизначный номер билета считается «счастливым», если сумма первых трех цифр номера равна сумме последних трех. Подсчитать количество счастливых билетов с номерами от M до N (000000 <= M <= N <= 999999).