Computer Programming

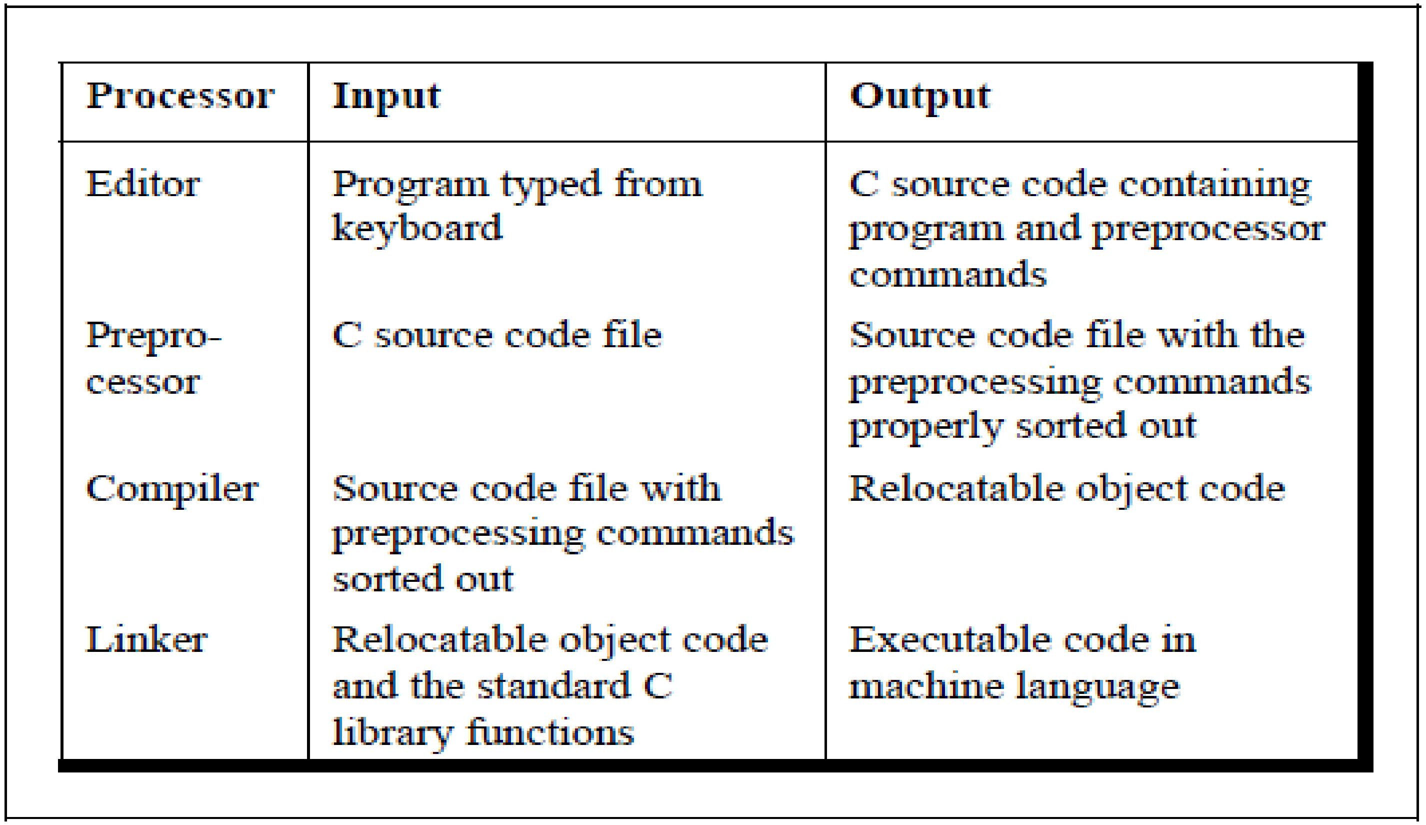
Sino-European Institute of Aviation Engineering

Module 7
The Preprocessor

Outline

Introduction

The role played by each processor program during the build process:



Processor	Input	Output
Editor	Program typed from keyboard	C source code containing program and preprocessor commands
Prepro-cessor	C source code file	Source code file with the preprocessing commands properly sorted out
Compiler	Source code file with preprocessing commands sorted out	Relocatable object code
Linker	Relocatable object code and the standard C library functions	Executable code in machine language

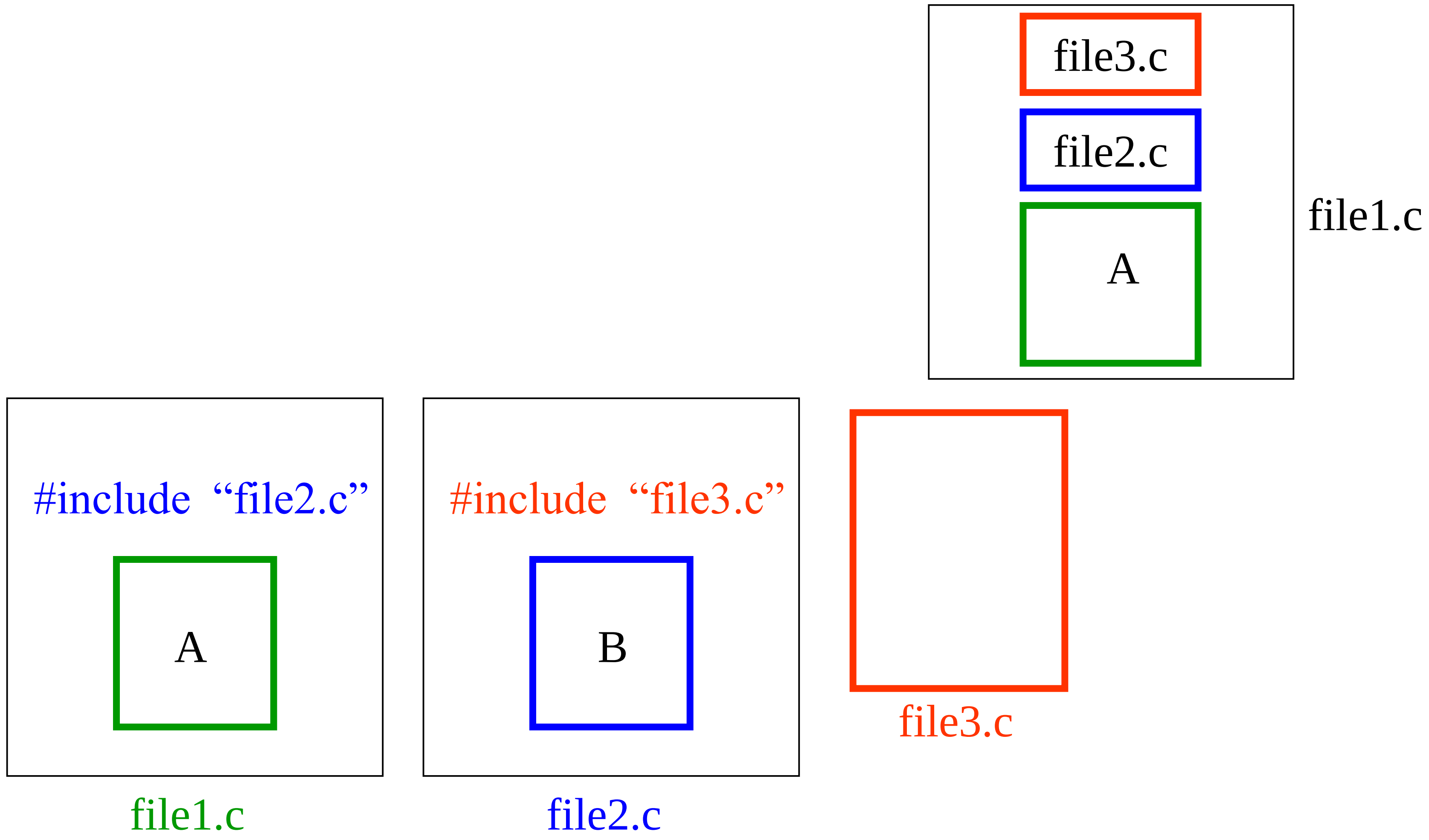
Introduction

- Preprocessing
  - Occurs before a program is compiled
  - Inclusion of other files
  - Definition of symbolic constants and macros
  - Conditional compilation of program code
  - Conditional execution of preprocessor directives
- Format of preprocessor directives
  - Lines begin with #
  - Only white space characters before directives on a line

The #include Preprocessor Directive

- #include
  - Copy of a specified file included in place of the directive
  - #include <filename>
    - Searches standard library for file
    - Use for standard library files
  - #include "filename"
    - Searches current directory, then standard library
    - Use for user-defined files
  - Used for:
    - Programs with multiple source files to be compiled together
    - Header file – has common declarations and definitions (classes, structures, function prototypes)
      - #include statement in each file

The #include Preprocessor Directive



The #Define Preprocessor Directive

- #define
  - Preprocessor directive used to create symbolic constants and macros
  - Symbolic constants
    - When program compiled, all occurrences of symbolic constant replaced with replacement text
  - Format
    - #define identifier replacement-text
    - Example: #define PI 3.14159
  - Cannot redefine symbolic constants once they have been created

The #Define Preprocessor Directive

```
#include <stdio.h>
#define PI 3.1415926
  void main()
    {float l,s,r,v;
     printf("input radius:");
     scanf("%f",&r);
     l=2.0*PI*r;
     s=PI*r*r;
     v=4.0/3*PI*r*r*r;
     printf("l=%10.4f\ns=%10.4f\nv=%10.4f\n",l,s,v);
}
```

The #Define Preprocessor Directive

- #define-macro
  - A macro without arguments is treated like a symbolic constant
  - A macro with arguments has its arguments substituted for replacement text, when the macro is expanded
  - Performs a text substitution – no data type checking
  - The macro

```
#define CIRCLE_AREA( x ) ( PI * ( x ) * ( x ) )
```

```
area = CIRCLE_AREA( 4 );
```

```
area = ( 3.14159 * ( 4 ) * ( 4 ) );
```

The #Define Preprocessor Directive

- Use parenthesis

```
#define CIRCLE_AREA( x )  PI * ( x ) * ( x )
```

```
area = CIRCLE_AREA( c + 2 );
        ↓
area = 3.14159 * c + 2 * c + 2;
```

- Multiple arguments

```
#define RECTANGLE_AREA( x, y )  ( ( x ) * ( y ) )
```

```
rectArea = RECTANGLE_AREA( a + 4, b + 7 );
        ↓
rectArea = ( ( a + 4 ) * ( b + 7 ) );
```

The #Define Preprocessor Directive

- Macros with arguments VS Function

```
#define  MAX(x,y)   (x)>(y)?(x):(y)
 …….
main()
{   int  a,b,c,d,t;
     …….
     t=MAX(a+b,c+d);
      ……
}
t=(a+b)>(c+d)?(a+b):(c+d);
```

```
int   max(int x,int y)
{  return(x>y?x:y);
}
main()
{     int a,b,c,d,t;
       …….
       t=max(a+b,c+d);
        ………
}
```

The #Define Preprocessor Directive

- Macros with arguments VS Function

	Macros with arguments	Function
When to process	Compile	Run
Argument type	No	Define formal and actual argument
process	Don't allot memory	Allot memory
Program length	Become longer	No change
Run speed	faster	slow

The #Define Preprocessor Directive

- #undef
  - Undefines a symbolic constant or macro
  - If a symbolic constant or macro has been undefined it can later be redefined

The #Define Preprocessor Directive

- Predefined symbolic constants
  - cannot be used in #define or #undef

Symbolic constant	Description
__LINE__	The line number of the current source code line (an integer constant).
__FILE__	The presumed name of the source file (a string).
__DATE__	The date the source file is compiled (a string of the form "Mmm dd yyyy" such as "Jan 19 2001").
__TIME__	The time the source file is compiled (a string literal of the form "hh:mm:ss").

The #Define Preprocessor Directive

(1) format.h

```
#include <stdio.h>
#define PR printf
#define NL "\n"
#define D "%d"
#define D1 D NL
#define D2 D D NL
#define D3 D D D NL
#define D4 D D D D NL
#define S "%s"
```

(2) file1.c

```
#include <stdio.h>
include  "format.h"
void main()
 { int a,b,c,d;
   char string[]="CHINA";
   a=1;b=2;c=3;d=4;
   PR(D1,a);
   PR(D2,a,b);
   PR(D3,a,b,c);
   PR(D4,a,b,c,d);
   PR(S,string);
 }
```

Conditional Compilation

- Conditional compilation
  - Control preprocessor directives and compilation
  - Cast expressions, sizeof, enumeration constants cannot be evaluated in preprocessor directives
  - Every #if must end with #endif
  - #ifdef short for #if defined(name)
  - #ifndef short for #if !defined(name)

Conditional Compilation

- Format

```
(1) # ifdef name
        statement₁
     # else
        statement₂
     # endif
```

```
(2) # ifndef name
        statement₁
     # else
        statement₂
     # endif
```

```
(3)  # if expression
          statement₁
       # else
          statement₁
      # endif
```

Conditional Compilation

```
#define R 1
main()
{  float c,r,s;
   printf ("input a number:");
   scanf("%f",&c);
   #if R
      r=3.14159*c*c;
      printf("area of round is: %f\n",r);
   #else
      s=c*c;
      printf("area of square is: %f\n",s);
   #endif
}
```

```
Input a number:3↙
area of round is: 28.274309
```

#define R 0

```
input a number:3↙
area of square is: 9.000000
```

Conditional Compilation

- Debugging

```
#define DEBUG  1
#ifdef DEBUG
      cerr << "Variable x = " << x << endl;
#endif
```

  - Defining DEBUG to 1 enables code
  - After code corrected, remove #define statement
  - Debugging statements are now ignored

Conditional Compilation

- Assert Macro
  - Header <assert.h>
  - Tests value of an expression, If 0 (false) prints error message and calls abort
  - Example:

```
assert( x <= 10 );
```

  - If NDEBUG is defined
    - All subsequent assert statements ignored

```
#define NDEBUG
```

Summary

- The preprocessor directives enable the programmer to write programs that are easy to develop, read, modify and transport to a different computer system.
- We can make use of various preprocessor directives such as #define, #include, #ifdef-#else-#endif.

Thank you!