Chapter 13

The Preprocessor

Chapter 13 - The Preprocessor

Outline

13.1 Introduction
13.2 The `#include` Preprocessor Directive
13.3 The `#define` Preprocessor Directive:
Symbolic Constants, Macros
13.5 Conditional Compilation

13.1 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 whitespace characters are allowed before directives on a line.

Preprocessor Directives

#define	#undef	#ifdef	#ifndef
#if	#else	#endif	#elif
#include	#error	#line	#pragma

13.2 The `#include` Preprocessor Directive

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

13.3 The #define Preprocessor Directive: Symbolic Constants

- Example: Following defines a hexadecimal constant

```
#define  SAYI  0x5F2B
```

- Cannot change or redefine symbolic constants once they have been created.

```
#define A  5
A = 8;          // Wrong
#define A  8    // Wrong
```

Suffixes for Integer and Floating-Point Constants

- C provides suffixes for constants
  - unsigned integer (u or U)
  - long integer (l or L)
  - unsigned long integer (ul, lu, UL or LU)
  - float (f or F)
  - long double (lf or LF)
  - Examples:

```
#define SABIT1  174u
#define SABIT2  467L
#define SABIT3  3452ul
```

  - If an integer constant is not suffixed, type determined by first type capable of storing a value of that size
    (Order: int, long int, unsigned long int)
  - If a floating point constant not suffixed, it is considered as `double`

13.4 The #define Preprocessor Directive: Macros

- Macro
  - Operation defined in `#define`
  - A macro without arguments is treated like a symbolic constant.
  - A macro with arguments is treated like a function. The arguments are substituted, when the macro is expanded.
  - Performs a text substitution – no data type checking
  - Recommended only for very short functions
  - The macro

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

would cause

```
area = CIRCLE_AREA( 4 ); // Original statement
```

to become

```
area = ( 3.14159 * ( 4 ) * ( 4 ) ); //Statement after preprocessing
```

13.4 The #define Preprocessor Directive: Macros

- Use parenthesis
  - Without them the macro

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

would cause

```
area = CIRCLE_AREA( c + 2 );
```

to become

```
area = 3.14159 * c + 2 * c + 2;
```

Wrong result!

- Multiple arguments

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

would cause

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

to become

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

13.4 The #undef Preprocessor Directive

- #undef
  - Undefines a symbolic constant or macro.
    (i.e. cancels a previosly defined constant or macro)
  - If a symbolic constant or macro has been undefined it can later be redefined again.

```
#include <stdio.h>
int main()
{
  #define A 5
  printf("%d", A);
  #undef A

  printf("%d", A);
}
```

Scope of A is limited between #define and #undef

Compiler no longer can recognize A here.

13.5 Conditional Compilation

- Conditional compilation
  - Control preprocessor directives and compilation
  - Cast expressions, `sizeof`, enumeration constants cannot be evaluated in preprocessor directives
  - Syntax similar to `if` statement

```
#if !defined( NULL )
   #define NULL 0
#endif
```

    - Determines if symbolic constant `NULL` has been defined
      - If `NULL` is defined, `defined( NULL )` evaluates to `1`
      - If `NULL` is not defined, this function defines `NULL` to be `0`
  - Every `#if` must end with `#endif`

13.5 Conditional Compilation

`#ifdef` short for `#if defined( name )`
`#ifndef` short for `#if !defined( name )`

- Other statements
  - `#else` – equivalent of `else` in an `if` statement
  - `#elif` – equivalent of `else if` in an `if` statement

Examples

```c
#include <stdio.h>
#include <stdlib.h>
#define SURUM 2

int main()
{
  printf("ÖRNEK PROGRAM \n");

#ifdef SURUM
  printf("Sürüm : %d ", SURUM);
#else
  printf("Sürüm :  Bilinmiyor");
#endif

  printf("\n");
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#define SURUM 2

int main()
{
  printf("ÖRNEK PROGRAM \n");

#if SURUM == 1
  printf("Sürüm : İlk");
#elif SURUM == 2
  printf("Sürüm : İkinci");
#else
  printf("Sürüm :  Bilinmiyor");
#endif

  printf("\n");
}
```

Chapter 14

Other C Topics

Chapter 14 - Other C Topics

Outline

14.2 Redirecting Input/Output on UNIX and DOS Systems
14.4 Using Command-Line Arguments
14.5 Compiling Multiple-Source-File Programs, C Project Files

Input/Output Redirection in Command-line

Input/Output Redirection

- Redirecting Input/Output on UNIX and DOS Systems
  - Standard input device is keyboard
  - Standard output device is screen
  - We can redirect input and output

14.2 Redirecting Input

- **Redirect input symbol (<)**
  - Operating system feature, not a C feature
  - Can be used both in UNIX and DOS
  - $ or % represents command line prompt symbol in Unix
  - Example:

```
$ myprog < myinput.txt
```

  - Rather than inputting values by hand from keyboard, program gets them from a file

14.2 Redirecting Output

- **Redirect output (>)**
  - Determines where output of a program goes
  - Example:

```
$ myprog > myout.txt
```

  - Instead of screen, output goes into `myout.txt` (erases previous contents if any)

14.2 Redirecting Both Input and Output

- Example:

```
$ myprog  <myinput.txt   >myout.txt
```

14.2 Redirecting and Appending Output

- **Append output (>>)**
  - Add output to end of file (preserve previous contents)
  - Example:

```
$ myprog >> myout.txt
```

  - Output is added onto the end of `myout.txt`

14.2 Piping Input and Output

- **Pipe command (|)**
  - Output of one program becomes input of another
  - Example:

```
$ prog1 | prog2
```

  - Output of `prog1` goes to `prog2`

Example: prog1.c

```
// Command-line pipe örneği:  prog1.exe  |  prog2.exe

#include <stdio.h>
int main()
{
  int i;
  for (i=1; i <=5; i++)
       printf("%d\n",i*10);
}
```

Example: prog2.c

```
// Command-line pipe örneği:  prog1.exe  |  prog2.exe

#include <stdio.h>
int main() {
  int i,num,Tot=0;

  for (i=1; i <=5; i++)  {
       printf("\n Enter a number :");
       scanf("%d",&num);
       Tot += num;
  }

  printf("Average is : %.2f\n", Tot/5.0);

}
```

Example: Piping two programs

```
$   prog1  |  prog2

Enter a number :
Enter a number :
Enter a number :
Enter a number :
Enter a number :
Average is : 30.00
```

Using Command-Line Arguments

14.4 Using Command-Line Arguments

- Pass arguments to `main` on DOS or UNIX
  - Define main as

    ```
    int main( int argc, char *argv[] )
    ```

  - `int argc`
    - Number of arguments passed
  - `char *argv[]`
    - Array of strings
    - Has names of arguments in order
      - `argv[ 0 ]` is first argument, which always contains the name of program

Example: carp.c

```c
#include <stdio.h>
#include <stdlib.h>    // atoi

int main (int argc, char *argv [ ] )
{
int Sayi1, Sayi2;
if ( argc != 3)
{
   printf("Yanlis sayida arguman girdiniz! \n");
   printf("Kullanim ornegi : Program-ismi  Sayi1  Sayi2 \n");
   system("pause");
   return 0;
}

Sayi1 = atoi( argv[1] );    // convert ascii string to integer
Sayi2 = atoi( argv[2] );    // convert ascii string to integer
printf("%d * %d = %d \n", Sayi1, Sayi2, Sayi1 * Sayi2);

} // end main
```

Example: Running program

- The following command-line instruction should be used to run the program.

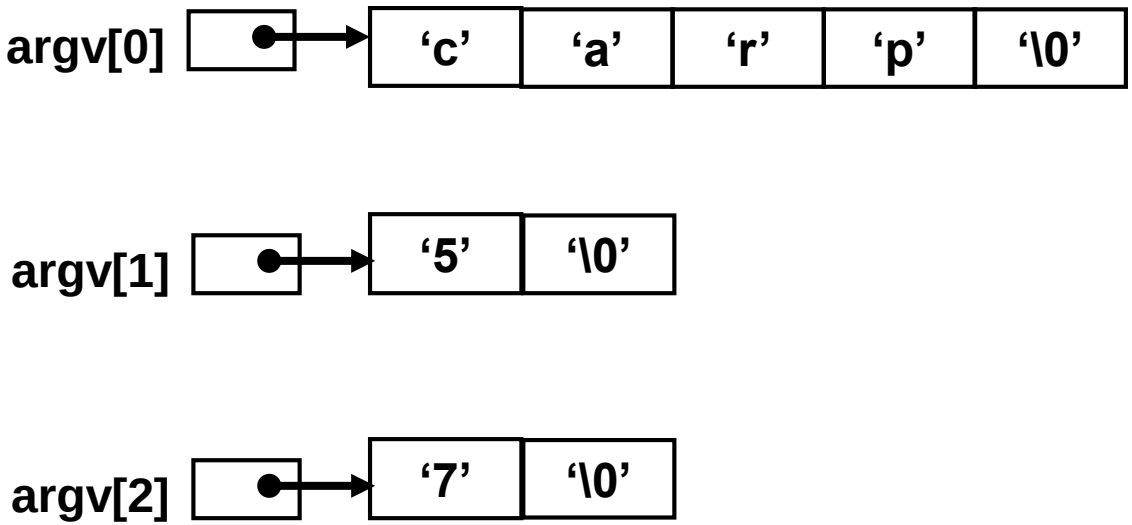
```
$ carp  5  7

5 * 7 = 35
```

- The arguments in main() are automatically assigned as the followings:

```
argc = 3
argv[0] = "carp“
argv[1] = "5“
argv[2] = "7"
```

Argument vector passed to main()

- argc is **<u>argument count</u>**.
- argv is an **<u>array of pointers</u>**.
- Each pointer points to a string.



Example: Alternative Method with sscanf

- Instead of atoi() functions, we can use sscanf() to get inputs from argv array.

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

int main (int argc, char *argv [ ] ) {
int Sayi1, Sayi2;

if ( argc != 3)
{
   printf("Yanlis sayida arguman girdiniz! \n");
   printf("Kullanim ornegi : Program-ismi  Sayi1  Sayi2 \n");
   system("pause");
   return 0;
}
   sscanf(argv[1], "%d" , &Sayi1);
   sscanf(argv[2], "%d" , &Sayi2);
   printf("%d * %d = %d \n", Sayi1, Sayi2, Sayi1 * Sayi2);

} // end main
```

Multiple Source File Programs

14.5 Compiling Multiple-Source-File Programs

- Programs with multiple source files
  - Function definition must be in one file (cannot be split up)
  - Global variables accessible to functions in same file
    - Global variables must be defined in every file in which they are used
  - Example:
    - If integer `sayi` is defined in one file
    - To use it in another file you must include the statement

```
extern int sayi;
```

  - `extern`
    - States that the variable is defined in another file
  - Function prototypes can be used in other files without an extern statement
    - Have a prototype in each file that uses the function

Example: Multiple-Source-Files and using extern

part1.c

```c
#include <stdio.h>

int main()
{
  extern int a;

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

part2.c

```c
// Global variable:

int a = 50;
```

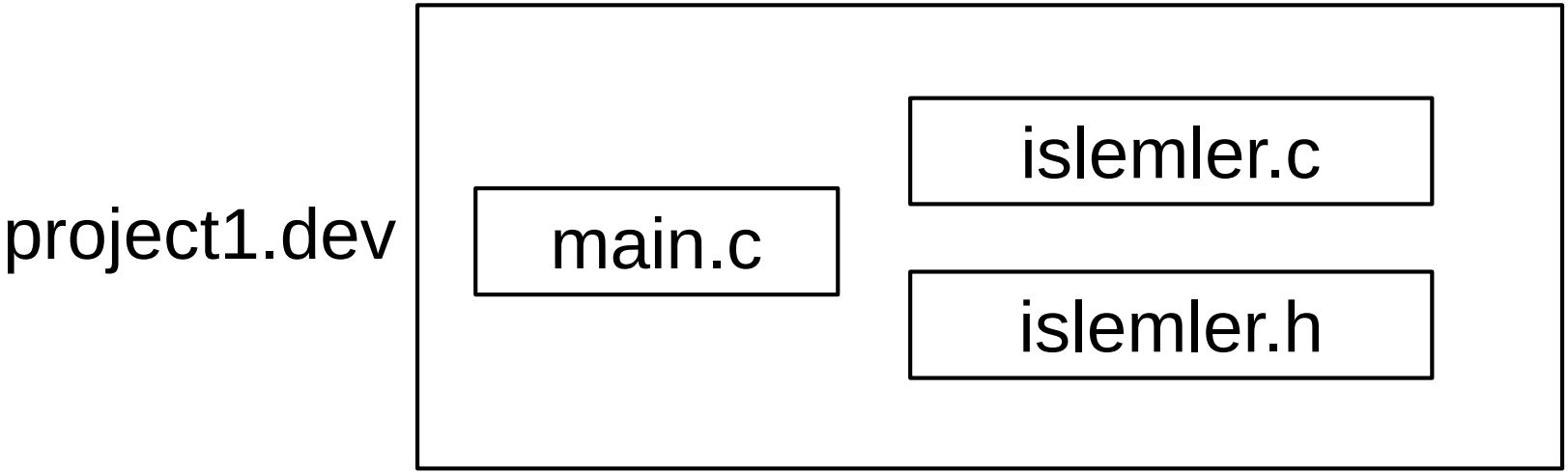
```
gcc  -o  myprog.exe  part1.c  part2.c
```

C Project Files

- A project file contains multiple C source files (*.c) and C header files (*.h) .
  - For Dev-C++, name of the project file can be anything, file type should be *"*.dev"* .
  - In Unix, name of the project file should be *"makefile"*.

- Only one file should contain the main() function.
- Other files can contain the functions written by the programmer.
- This method is useful when the source code is too long.

Example: Project File



- The following command-line instruction can be used to compile and link the multiple C source files.

```
gcc -o project1.exe main.c islemler.c
```

main.c

```c
#include <stdio.h>
#include "islemler.h"

int main() {
  int cevap;
  while(1) { // Infinite loop
     printf("ISLEMLER \n");
     printf("1. Notlari gir \n");
     printf("2. Artan sirada goruntule \n");
     printf("3. Azalan sirada goruntule \n");
     printf("4. Ortalama \n");
     printf("5. Cikis \n");
     printf("   Seciminiz : ");
     scanf("%d", &cevap);
     switch (cevap) {
       case 1: notgir();       break;
       case 2: sirala(ARTAN);  break;
       case 3: sirala(AZALAN); break;
       case 4: ortalama();     break;
       case 5: return 0;       //Stop
       default: printf(" Gecersiz secenek \n");
     } // end switch
   } // end while
   printf(" Program bitti \n");
} // end main
```

islemler.h

```
//islemler.h

//Global definitions:
#define ARTAN  1
#define AZALAN 2


// Function prototypes:
void notgir();
void sirala(int );
void ortalama();
```

islemler.c

```c
#include <stdio.h>
#include "islemler.h"

int notlar[50];
int sayac; // Ogrenci sayisi
void swap(int *a, int *b); //Prototype


void notgir()
{
 printf("Notlari girin (sonlandirmak icin -1) : ");
 sayac=0;
 do
 {
   scanf("%d", &notlar[sayac]);
   sayac++;
 } while (notlar[sayac-1] != -1);

 sayac--;
} // end notgir
```

islemler.c (cont.)

```
void sirala(int siralama_yonu)
{
 int i,j;
 //Selection Sort method:
 for (i=0; i < sayac-1; i++)
     for (j=i+1; j < sayac; j++)
         if (siralama_yonu == ARTAN) //Sort by ascending order
         {
            if (notlar[i] > notlar[j])
                swap(&notlar[i], &notlar[j]);
         }
         else //Sort by descending order
         {
            if (notlar[i] < notlar[j])
                swap(&notlar[i], &notlar[j]);
         }
 // Diziyi ekrana yaz:
 for (i=0; i < sayac; i++)
     printf("%d \n", notlar[i]);
} // end sirala
```

islemler.c (cont.)

```
void swap(int *a, int *b)
{
  int tmp;

  tmp = *a;
  *a = *b;
  *b = tmp;
}
```

```
void ortalama()
{
 int i, Tot=0;

 for (i=0; i < sayac; i++)
     Tot += notlar[i];

 printf("Ortalama = %f \n",
        (float) Tot / sayac);
}
```

END OF SLIDES