CPE100
Computer Programming for Engineers

MODULE 3
Introduction to C Programming

Assc. Prof. Dr. Natasha Dejdumrong

KING MONGKUT'S UNIVERSITY
OF TECHNOLOGY THONBURI

Outlines

C Simple Programs

Memory Concept

Arithmetic Operators

Relational Operators

C Conditional Statements

A Simple C Program: Print a Line of Text

```c
// Fig. 2.1: fig02_01.c
// A first program in C.
#include <stdio.h>

// function main begins program execution
int main( void )
{
   printf( "Welcome to C!\n" );
} // end function main
```

Comments

Preprocessor Directive

Blank line or white space

main Function

Output statement

```
Welcome to C!
```

Fig. 2.1 | A first program in C.

Escape Sequence

Escape sequence	Description
\n	Newline. Position the cursor at the beginning of the next line.
\t	Horizontal tab. Move the cursor to the next tab stop.
\a	Alert. Produces a sound or visible alert without changing the current cursor position.
\\	Backslash. Insert a backslash character in a string.
\"	Double quote. Insert a double-quote character in a string.

Fig. 2.2 | Some common escape sequences .

Using Multiple printf

```c
// Fig. 2.3: fig02_03.c
// Printing on one line with two printf statements.
#include <stdio.h>

// function main begins program execution
int main( void )
{
   printf( "Welcome " );
   printf( "to C!\n" );
} // end function main
```

```
Welcome to C!
```

Fig. 2.3 | Printing on one line with two `printf` statements.

Using Multiple printf

```c
// Fig. 2.4: fig02_04.c
// Printing multiple lines with a single printf.
#include <stdio.h>

// function main begins program execution
int main( void )
{
   printf( "Welcome\nto\nC!\n" );
} // end function main
```

```
Welcome
to
C!
```

Fig. 2.4 | Printing multiple lines with a single `printf`.

Another Simple C Program:Adding 2 Integers

```c
// Fig. 2.5: fig02_05.c
// Addition program.
#include <stdio.h>

// function main begins program execution
int main( void )
{
   int integer1; // first number to be entered by user
   int integer2; // second number to be entered by user
   int sum; // variable in which sum will be stored

   printf( "Enter first integer\n" ); // prompt
   scanf( "%d", &integer1 ); // read an integer

   printf( "Enter second integer\n" ); // prompt
   scanf( "%d", &integer2 ); // read an integer

   sum = integer1 + integer2; // assign total to sum

   printf( "Sum is %d\n", sum ); // print sum
} // end function main
```

Variables and Variable definition

```c
int integer1, integer2, sum;
```

Prompting Message

scanf Function

Assignment Statement

```c
printf( "Sum is %d\n", integer1 + integer2 );
```

Another Simple C Program:Adding 2 Integers

```c
// Fig. 2.5: fig02_05.c
// Addition program.
#include <stdio.h>

// function main begins program execution
int main( void )
{
   int integer1; // first number to be entered by user
   int integer2; // second number to be entered by user
   int sum; // variable in which sum will be stored

   printf( "Enter first integer\n" ); // prompt
   scanf( "%d", &integer1 ); // read an integer

   printf( "Enter second integer\n" ); // prompt
   scanf( "%d", &integer2 ); // read an integer

   sum = integer1 + integer2; // assign total to sum

   printf( "Sum is %d\n", sum ); // print sum
} // end function main
```

```
Enter first integer
45
Enter second integer
72
Sum is 117
```

Fig. 2.5 | Addition program.

Memory Concepts

Variable names such as `integer1`, `integer2` and `sum` actually correspond to locations in the computer's memory. Every variable has a name, a **type** and a **value**.

In the addition program of Fig. 2.5, when the statement (line 13)

```
scanf( "%d", &integer1 ); // read an integer
```

is executed, the value entered by the user is placed into a memory location to which the name `integer1` has been assigned. Suppose the user enters the number `45` as the value for `integer1`. The computer will place `45` into location `integer1`, as shown in Fig. 2.6.

Fig. 2.6 | Memory location showing the name and value of a variable.

Memory Concepts

Once the program has obtained values for `integer1` and `integer2`, it adds these values and places the total into variable `sum`. The statement (line 18)

```
sum = integer1 + integer2; // assign total to sum
```

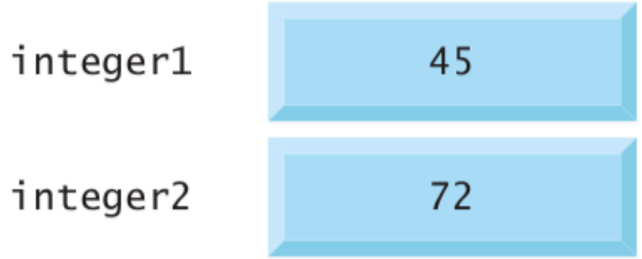
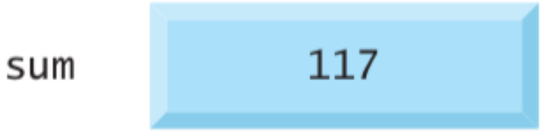


Fig. 2.7 | Memory locations after both variables are input.



Arithmetic Operators in C

C operation	Arithmetic operator	Algebraic expression	C expression
Addition	+	$f + 7$	`f + 7`
Subtraction	−	$p - c$	`p - c`
Multiplication	*	bm	`b * m`
Division	/	x / y or $\frac{x}{y}$ or $x \div y$	`x / y`
Remainder	%	$r \bmod s$	`r % s`

Fig. 2.9 | Arithmetic operators.

Rules of Operator Precedence

Operator(s)	Operation(s)	Order of evaluation (precedence)
( )	Parentheses	Evaluated first. If the parentheses are nested, the expression in the *innermost* pair is evaluated first. If there are several pairs of parentheses "on the same level" (i.e., not nested), they're evaluated left to right.
* / %	Multiplication Division Remainder	Evaluated second. If there are several, they're evaluated left to right.
+ -	Addition Subtraction	Evaluated third. If there are several, they're evaluated left to right.
=	Assignment	Evaluated last.

Fig. 2.10 | Precedence of arithmetic operators.

Sample Algebraic and C Expression

Algebra: $m = \frac{a + b + c + d + e}{5}$

C: `m = ( a + b + c + d + e ) / 5;`

The parentheses are required to group the additions because division has higher precedence than addition. The entire quantity `( a + b + c + d + e )` should be divided by `5`. If the parentheses are erroneously omitted, we obtain `a + b + c + d + e / 5`, which evaluates incorrectly as

$$a + b + c + d + \frac{e}{5}$$

Sample Algebraic and C Expression

The following expression is the equation of a straight line:

Algebra: $y = mx + b$

C: `y = m * x + b;`

No parentheses are required. The multiplication is evaluated first because multiplication has a higher precedence than addition.

The following expression contains remainder (%), multiplication, division, addition, subtraction and assignment operations:

Algebra: $z = pr\%q + w/x - y$

C: `z = p * r % q + w / x - y;`

6 1 2 4 3 5

Evaluation of a Second-Degree Polynomial

To develop a better understanding of the rules of operator precedence, let's see how C evaluates a second-degree polynomial.

```
y = a * x * x + b * x + c;
    6   1   2   4   3   5
```

The circled numbers under the statement indicate the order in which C performs the operations. There's no arithmetic operator for exponentiation in C, so we've represented x^2 as `x * x`. The C Standard Library includes the `pow` ("power") function to perform exponentiation. Because of some subtle issues related to the data types required by `pow`, we defer a detailed explanation of `pow` until Chapter 4.

Evaluation of a Second-Degree Polynomial

Step 1.	`y = 2 * 5 * 5 + 3 * 5 + 7;` `2 * 5 is 10`	*(Leftmost multiplication)*
Step 2.	`y = 10 * 5 + 3 * 5 + 7;` `10 * 5 is 50`	*(Leftmost multiplication)*
Step 3.	`y = 50 + 3 * 5 + 7;` `3 * 5 is 15`	*(Multiplication before addition)*
Step 4.	`y = 50 + 15 + 7;` `50 + 15 is 65`	*(Leftmost addition)*
Step 5.	`y = 65 + 7;` `65 + 7 is 72`	*(Last addition)*
Step 6.	`y = 72`	*(Last operation—place 72 in y)*

Relational Operators in C

Algebraic equality or relational operator	C equality or relational operator	Example of C condition	Meaning of C condition
Equality operators			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y
Relational operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y

Fig. 2.12 | Equality and relational operators.

Relational Operators in C

```c
// Fig. 2.13: fig02_13.c
// Using if statements, relational
// operators, and equality operators.
#include <stdio.h>

// function main begins program execution
int main( void )
{
   int num1; // first number to be read from user
   int num2; // second number to be read from user

   printf( "Enter two integers, and I will tell you\n" );
   printf( "the relationships they satisfy: " );

   scanf( "%d%d", &num1, &num2 ); // read two integers

   if ( num1 == num2 ) {
      printf( "%d is equal to %d\n", num1, num2 );
   } // end if

   if ( num1 != num2 ) {
      printf( "%d is not equal to %d\n", num1, num2 );
   } // end if

   if ( num1 < num2 ) {
      printf( "%d is less than %d\n", num1, num2 );
   } // end if

   if ( num1 > num2 ) {
      printf( "%d is greater than %d\n", num1, num2 );
   } // end if

   if ( num1 <= num2 ) {
      printf( "%d is less than or equal to %d\n", num1, num2 );
   } // end if

   if ( num1 >= num2 ) {
      printf( "%d is greater than or equal to %d\n", num1, num2 );
   } // end if
} // end function main
```

```
Enter two integers, and I will tell you
the relationships they satisfy: 3 7
3 is not equal to 7
3 is less than 7
3 is less than or equal to 7
```

```
Enter two integers, and I will tell you
the relationships they satisfy: 22 12
22 is not equal to 12
22 is greater than 12
22 is greater than or equal to 12
```

```
Enter two integers, and I will tell you
the relationships they satisfy: 7 7
7 is equal to 7
7 is less than or equal to 7
7 is greater than or equal to 7
```

Precedence in C

Operators				Associativity
()				left to right
*	/	%		left to right
+	-			left to right
<	<=	>	>=	left to right
==	!=			left to right
=				right to left

Fig. 2.14 | Precedence and associativity of the operators discussed so far.

Keywords in C

Keywords			
auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Secure in C Programming

Avoid Single-Argument **`printf`**s[3]

One such guideline is to *avoid using* `printf` *with a single string argument*. If you need to display a string that *terminates with a newline*, use the **`puts` function**, which displays its string argument followed by a newline character. For example, in Fig. 2.1, line 8

```
printf( "Welcome to C!\n" );
```

should be written as:

```
puts( "Welcome to C!" );
```

We did not include `\n` in the preceding string because `puts` adds it automatically.

Secure in C Programming

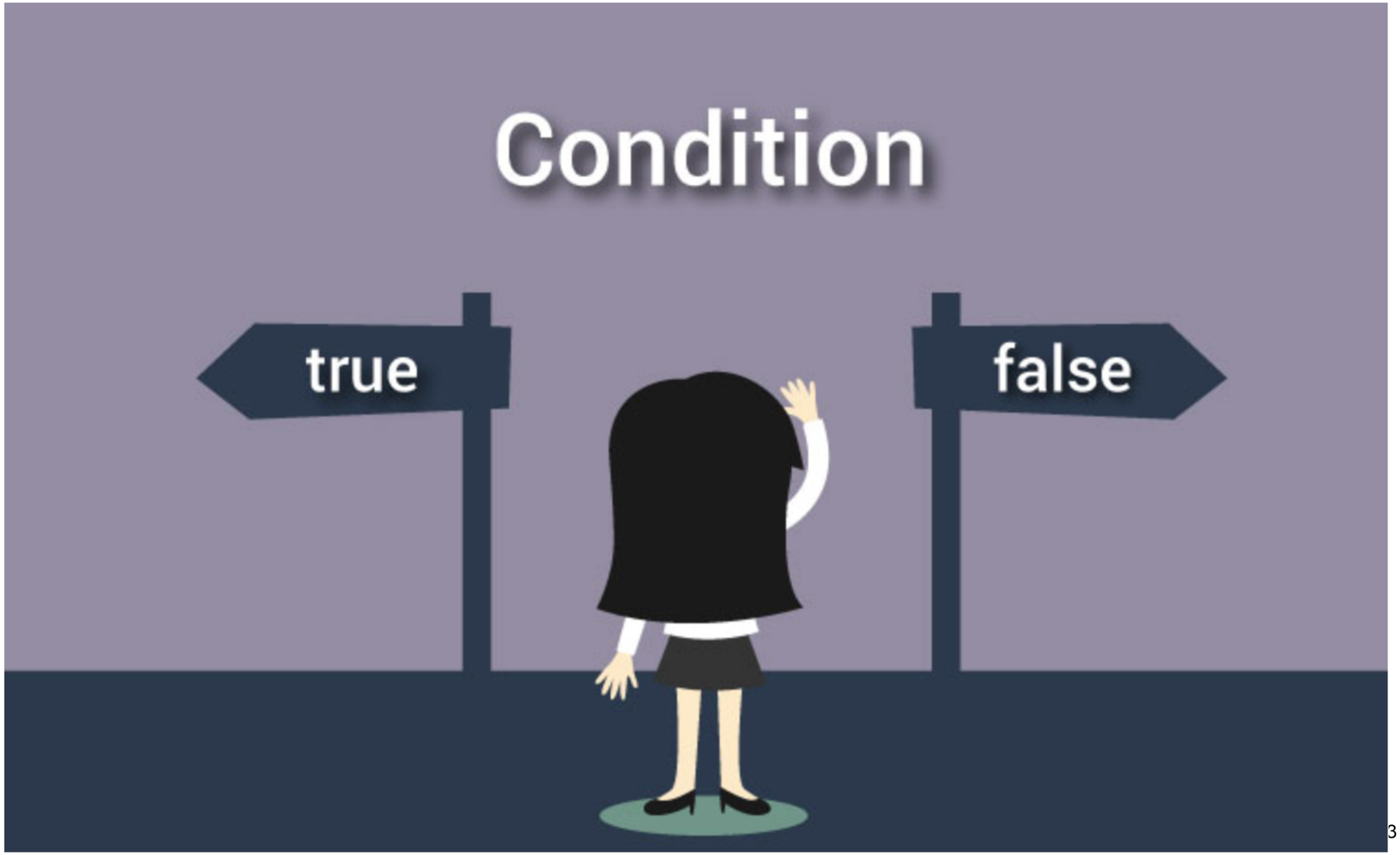
If you need to display a string *without* a terminating newline character, use `printf` with *two* arguments—a "%s" format control string and the string to display. The **%s conversion specifier** is for displaying a string. For example, in Fig. 2.3, line 8

```
printf( "Welcome " );
```

should be written as:

```
printf( "%s", "Welcome " );
```

C Conditional Statements



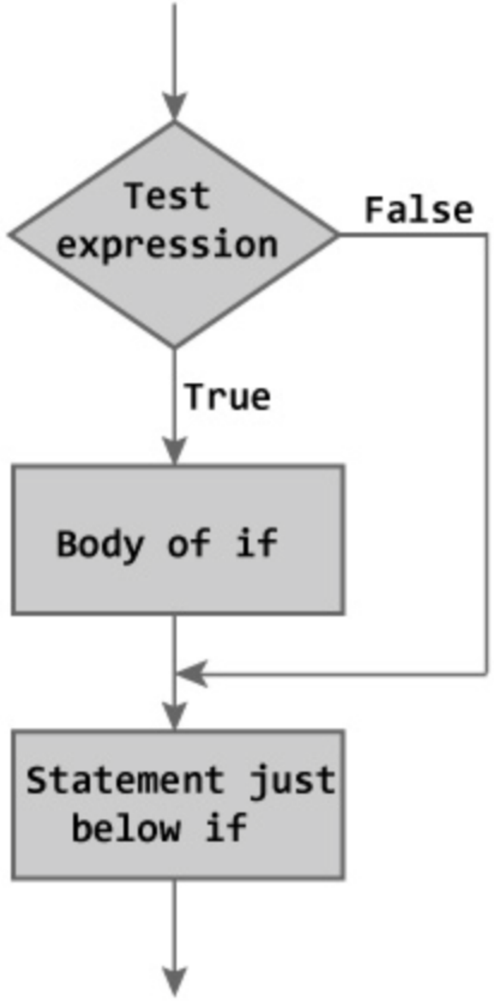
C if statement

```
if (testExpression)
{
    // statements
}
```

C if statement

```
if (testExpression)
{
    // statements
}
```

Flowchart of if statement



Example #1: C if statement

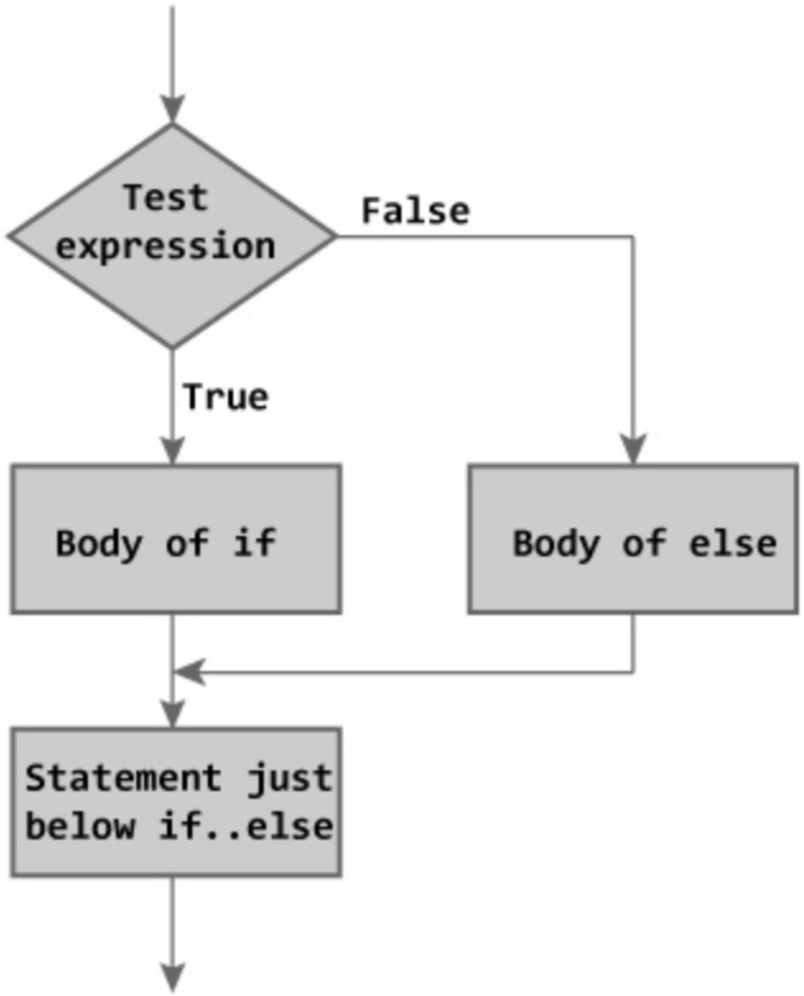
```c
// Program to display a number if user enters negative number
// If user enters positive number, that number won't be displayed

#include <stdio.h>
int main()
{
     int number;

     printf("Enter an integer: ");
     scanf("%d", &number);

     // Test expression is true if number is less than 0
     if (number < 0)
     {
          printf("You entered %d.\n", number);
     }

     printf("The if statement is easy.");

     return 0;
}
```

C if...else statement

```
if (testExpression) {
     // codes inside the body of if
}
else {
     // codes inside the body of else
}
```

C if...else statement

```
if (testExpression) {
     // codes inside the body of if
}
else {
     // codes inside the body of else
}
```



Example #2: C if...else statement

```c
// Program to check whether an integer entered by the user is odd or even

#include <stdio.h>
int main()
{
     int number;
     printf("Enter an integer: ");
     scanf("%d",&number);

     // True if remainder is 0
     if( number%2 == 0 )
          printf("%d is an even integer.",number);
     else
          printf("%d is an odd integer.",number);
     return 0;
}
```

Nested if...else statement

```
if (testExpression1)
{
    // statements to be executed if testExpression1 is true
}
else if(testExpression2)
{
    // statements to be executed if testExpression1 is false and testExpression2 :
}
else if (testExpression 3)
{
    // statements to be executed if testExpression1 and testExpression2 is false a
}
.
.
else
{
    // statements to be executed if all test expressions are false
}
```

Example #3: C Nested if...else

```c
// Program to relate two integers using =, > or <

#include <stdio.h>
int main()
{
     int number1, number2;
     printf("Enter two integers: ");
     scanf("%d %d", &number1, &number2);

     //checks if two integers are equal.
     if(number1 == number2)
     {
          printf("Result: %d = %d",number1,number2);
     }

     //checks if number1 is greater than number2.
     else if (number1 > number2)
     {
          printf("Result: %d > %d", number1, number2);
     }

     // if both test expression is false
     else
     {
          printf("Result: %d < %d",number1, number2);
     }

     return 0;
}
```

Question and Answer