Introduction to C Programming

Section 2

OBJECTIVES

- Simple C program
  - Variables
  - Data types
  - Arithmetic in C
  - Operations

A Simple C Program

- Examples:

```
// int is return data type
// main is entrance function
int main()
{
    statement 1;
    statement 1;
    // ....
    return 0;
}
```

C Program

- Examples:
  - Header file
  - Main function
  - Variables
  - Input and output
  - Process

```c
#include <stdio.h>  // header file (preprocessor )
//  calculating sum of two  user input  variables
int main()
{
    /* variable definition */
    int a;
    int b;
    int result = 0;
    // get first variables form user
    printf("Enter first number:\n");
    scanf("%d", &a);
    // get scoend variables form user
    printf("Enter scoend number:\n");
    scanf("%d", &b);
    //  sum of input variables
    result = a + b;
    printf("%d + %d = %d\n", a, b, result);
    system("Pause");
    return 0;
}
```

* VIM default coloring for C

A Simple C Program: Printing a Line of Text

```
1  /* Fig. 2.1: fig02_01.c
2     A first program in C */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main( void )
7  {
8     printf( "Welcome to C!\n" );
9
10    return 0; /* indicate that program ended successfully */
11 } /* end function main */
```

```
Welcome to C!
```

Fig. 2.1 | A first program in C.

A Simple C Program: Printing a Line of Text

- When encountering a backslash in a string, the compiler looks ahead at the next character and combines it with the backslash to form an escape sequence.
- The escape sequence `\n` means newline.

A Simple C Program: Printing a Line of Text

- Line 10
  - `return 0; /* indicate that program ended successfully */`
- is included at the end of every `main` function.
- The keyword `return` is one of several means we'll use to exit a function.
- When the return statement is used at the end of `main` as shown here, the value `0` indicates that the program has terminated successfully.

Variables

- Have the same meaning as variables in algebra
  - Single alphabetic character
  - Each variable needs an identifier that distinguishes it from the others
    - a = 5
    - x = a + b
- valid identifier in C may be given representations containing multiple characters
  - A-Z, a-z, 0-9, and _ (underscore character)
  - First character must be a letter or underscore (no, _no 9no)
  - Usually only the first 32 characters are significant
  - There can be no embedded blanks (student no)
  - Identifiers are case sensitive (area, Area, AREA, ArEa)
  - Keywords cannot be used as identifiers

Reserved Words (Keywords) in C

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

Error-Prevention Tip 2.1

Avoid starting identifiers with the underscore character (_) to prevent conflicts with compiler-generated identifiers and standard library identifiers.

Variable declaration

- Before using a variable, you must declare it
- All variables must be defined with a name and a data type.
  - Data_Type Identifier;
    - int width; // width of rectangle
    - float area; // result of calculating area stored in it
    - char separator; // word separator
  - Data_Type Identifier = Initial_Value;
    - int width = 10; // width of rectangle
    - float area = 255; // result of calculating area stored in it
    - char seperator = ‘,’; // word separator
  - Data_Type Identifier, Identifier, Identifier;
    - int width, length, temporary;
    - float radius, area = 0;

Data types

- Minimal set of basic data types
  - primitive data types
    - int
    - float
    - double
    - char
    - Void
- The size and range of these data types may vary among processor types and compilers

Another Simple C Program: Adding Two 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 input by user  */
   int integer2; /* second number to be input 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 */

   return 0; /* indicate that program ended successfully */
} /* end function main */
```

Fig. 2.5 | Addition program. (Part 1 of 2.)

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

Fig. 2.5 | Addition program. (Part 2 of 2.)

Type	Storage size	Value range	format specifier
char	1 byte	-128 to 127 or 0 to 255	%c
unsigned char	1 byte	0 to 255	%c
signed char	1 byte	-128 to 127	%c
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647	d%
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295	%u
short	2 bytes	-32,768 to 32,767	%hd
unsigned short	2 bytes	0 to 65,535	%hu
long	8 bytes	-9223372036854775808 to 9223372036854775807	%ld
unsigned long	8 bytes	0 to 18446744073709551615	%lu
float	4		%f
double	8		%lf
long double	12		%lf

Printf / Scanf

- prints the literal `Enter first integer` on the screen and positions the cursor to the beginning of the next line.
  - `printf( "Enter first integer\n" ); /* prompt */`
- This message is called a prompt because it tells the user to take a specific action.
- The `scanf` function reads from the standard input, which is usually the keyboard.
  - `scanf( "%d", &integer1 ); /* read an integer */`
- uses `scanf` to obtain a value from the user.

Printf / Scanf

- This `scanf` has two arguments, `"%d"` and `&integer1`.
- The first argument, the format control string, indicates the type of data that should be input by the user.
- The `%d` conversion specifier indicates that the data should be an integer (the letter `d` stands for “decimal integer”).
- The second argument of `scanf` begins with an ampersand (&)—called the address operator in C—followed by the variable name. The ampersand, when combined with the variable name, tells `scanf` the location (or address) in memory at which the variable `integer1` is stored.

Arithmetic in C

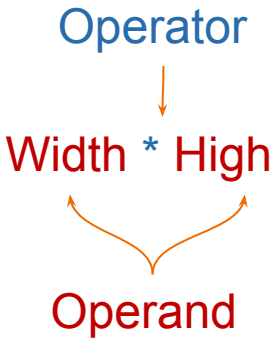
- The C arithmetic operators are summarized in Fig. 2.9.

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.

Operators

- Arithmetic Operators
  - unary operators
    - operators that require only one operand
  - binary operators
    - operators that require two operands
- Equality and Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Conditional Operator



Arithmetic Operators

- Unary Operator

C operation	Operator	Expression	Explanation
Positive	+	a = +3;	
Negative	-	b = -4;	
Increment	++	i++;	Equivalent to i = i + 1
Decrement	- -	i - -;	Equivalent to i = i - 1

PRE / POST Increment

- Consider this example:

```
int width = 9;
printf("%d\n", width++);
printf("%d\n", width);
```

- But if we have:

```
int width = 9;
printf("%d\n", ++width);
printf("%d\n", width);
```

PRE / POST Increment

- Consider this example:

```
int width = 9;
printf("%d\n", width++);
printf("%d\n", width);
```

```
int width = 9;
printf("%d\n", width);
width++;
printf("%d\n", width);
```

- But if we have:

```
int width = 9;
printf("%d\n", ++width);
printf("%d\n", width);
```

```
int width = 9;
width++;
printf("%d\n", width);
printf("%d\n", width);
```

PRE / POST Increment

- Consider this example:

```
int width = 9;
printf("%d\n", width++);
printf("%d\n", width);
```

```
int width = 9;
printf("%d\n", width);
width++;
printf("%d\n", width);
```

```
9
10
```

- But if we have:

```
int width = 9;
printf("%d\n", ++width);
printf("%d\n", width);
```

```
int width = 9;
width++;
printf("%d\n", width);
printf("%d\n", width);
```

```
10
10
```

PRE / POST Increment

```
int R = 10;
int count = 10;
```

++ Or -- Statement	Equivalent Statements	R	count
R = count++;	R = count; count = count + 1;	10	11
R = ++count;	count = count + 1; R = count;	11	11
R = count--;	R = count; count = count – 1;	10	9
R = --count;	count = count – 1; R = count;	9	9

Arithmetic Operators

- Binary Operators

C operation	Operator	Expression
Addition	+	b = a + 3;
Subtraction	-	b = a – 4;
Multiplication	*	b = a * 3;
Division	/	b = a / c;
Modulus (integer)	%	b = a % c;

Division

- The division of variables of type integer will always produce a variable of type integer as the result

- Example

```
int a = 7, b;
float z;
b = a / 2;
z = a / 2.0;
printf("b = %d, z = %f\n", b, z);
```

Since b is declared as an integer, the result of a/2 is 3, not 3.5

```
b = 3, z = 3.500000
```

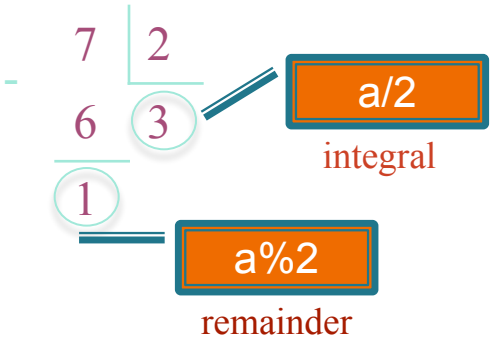
Modulus

- You could only use modulus (%) operation on integer variables (int, long, char)
  - z = a % 2.0; // error
  - z = a % 0; // error

- Example

```
int a = 7, b, c;
b = a % 2;
c = a / 2;
printf("b = %d\n", b);
printf("c = %d\n", c);
```

Modulus will result in the remainder of a/2.



Equality and Relational Operators

- Equality Operators:

Operator	Example	Meaning
==	x == y	x is equal to y
!=	x != y	x is not equal to y

- Relational Operators:

Operator	Example	Meaning
>	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

Bitwise Operators

Operator	Name	Description
&	AND	Result is 1 if both operand bits are 1
|	OR	Result is 1 if either operand bit is 1
^	XOR	Result is 1 if operand bits are different
~	Not (Ones Complement)	Each bit is reversed
<<	Left Shift	Multiply by 2
>>	Right Shift	Divide by 2

Logical Operators

- Logical operators are useful when we want to test multiple conditions
  - AND
  - OR
  - NOT

&& - Logical AND

- All the conditions must be true for the whole expression to be true
  - Example: if (a == 1 && b == 2 && c == 3)
    - means that the if statement is only true when a == 1 and b == 2 and c == 3

e1	e2	Result = e1 && e2
0	0	0
0	1	0
1	0	0
1	1	1

e1	e2	Result = e1 && e2
false	false	false
false	true	false
true	false	false
true	true	true

|| - Logical OR

- The truth of **one** condition is enough to make the whole expression true
- Example: if (a == 1 || b == 2|| c == 3)
  - means the if statement is true when either one of a, b or c has the right value

e1	e2	Result = e1 || e2
0	0	0
0	1	1
1	0	1
1	1	1

e1	e2	Result = e1 || e2
false	false	false
false	true	true
true	false	true
true	true	true

! - Logical NOT

- Reverse the meaning of a condition
- Example: if (!(radius > 90))
  - Means if radius not bigger than 90.

e1	Result = !e1
0	1
0	1
1	0
1	0

e1	Result = !e1
false	true
false	true
true	false
true	false

Conditional Operator

- The conditional operator (?:) is used to simplify an if/else statement
  - Condition ? Expression1 : Expression2;

- The statement above is equivalent to:

```
if (Condition)
    Expression1;
else
    Expression2;
```

- Which are more readable?

Conditional Operator

- Example:

if/else statement:

```
if (total > 12)
    grade = ‘P’;
else
    grade = ‘F’;
```

conditional statement:

```
(total > 12) ? grade = ‘P’: grade = ‘F’;
```

OR

```
grade =( total > 12) ? ‘P’: ‘F’;
```