Introduction to Scientific and Engineering Computation (BIL 104E)

Lecture 2

Structure of C programming

A Simple C Program: Printing a Line of Text

Program Line Number

```
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!
```

The C language does not have line numbers at all.

This is only done here as a reference for when we discuss what each line of a program does

C syntax

```
/*  comments  */

/* comments
        …
        …          */
```

Line 1 and 2:

```
1   /* Fig. 2.1: fig02_01.c
2      A first program in C */
```

This line starts with a combination of slash and asterisk , /* , and ends with */.

A Simple C Program: Printing 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" );

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

```
Welcome to C!
```

The C compilers ignores everyting between comment marks

Comments improve readibility of the program, hence other people read and understand easily your program.

Comments should be simple and give some explanation about code part.

A Simple C Program: Printing a Line of Text

```c
 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!
```

Line 3 tells the preprocessor to include the contents of the **standart input /output header** (<stdio.h>)

Pound sign adds header file to the preprocessor step.

C syntax

# include <header_file>

Header file contains information about how to work some functions such as **printf**

A Simple C Program: Printing a Line of Text

Execution starts with main function.

```
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!
```

Line 6 `int main( void )` is a part of every C program.
C programs contain one or more functions, one of which must be **main.**

C syntax

returns_type function_name(information received by function)

int indicates that **main** returns an integer (whole number) value.

void indicates that **main** does not receive any information.

A Simple C Program: Printing a Line of Text

Body of the **main** function begins with a **left brace** and ends with a corresponding **right brace**

```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" );

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

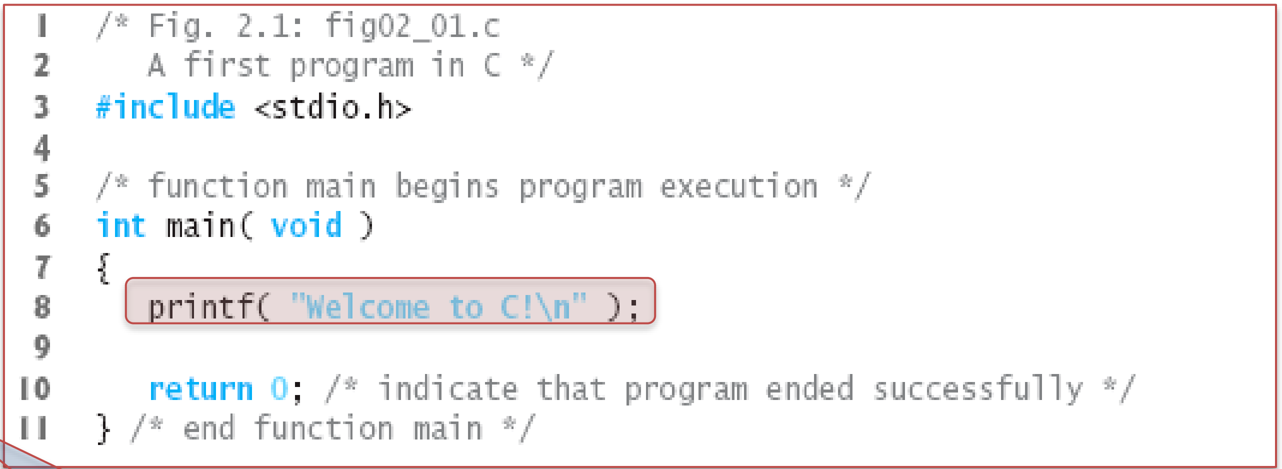
```
Welcome to C!
```

C syntax

returns_type   function_name(information received by function)
{
            body of function;
}

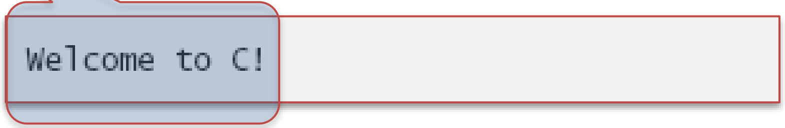
This is a statement and every statement must end with a semicolon (;)

A Simple C Program: Printing 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" );

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

- Character string,
- Message
- literal



```
Welcome to C!
```

printf instructs the computer to perform an action.
The action for **printf** is to print on the screen the **string** of characters marked by the **quotation** marks **double quotes** (“ character string “).

Notice that the characters **\n** were not printed on the screen.
The **backslash** (****) is called an **escape character**.
The combination of backslash and next character forms **escape sequence.**
The **escape sequence** **\n** means **newline** that’s why cursor moves **newline** after **character string**.

A Simple C Program: Printing a Line of Text

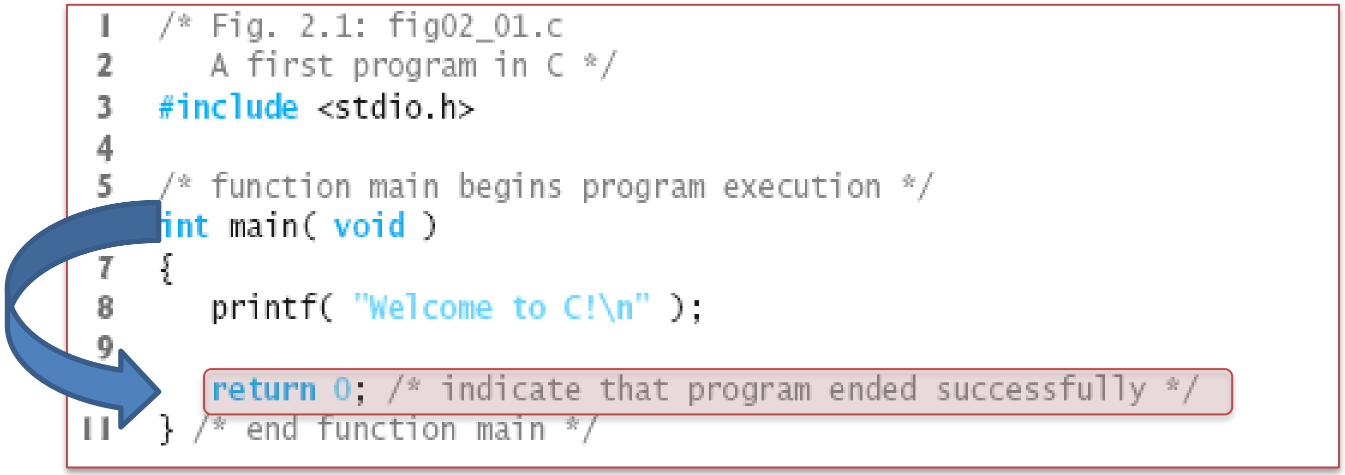
```c
/* Fig. 2.1: fig02_01.c
   A first program in C */
#include <stdio.h>

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

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

```
Welcome to C!
```

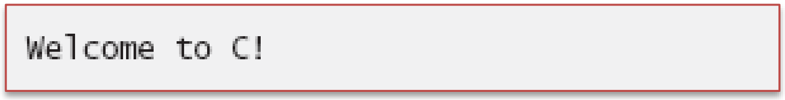
Some common escape sequences

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. Sound the system bell.
\\	Backslash. Insert a backslash character in a string.
\"	Double quote. Insert a double-quote character in a string.

A Simple C Program: Printing 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" );

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



```
Welcome to C!
```

main function is supposed to return integer value (**int**) and the program has terminated successfully through `return 0`.

Standart library functions like **printf** and **scanf** are not part of the C programming language, that's why compiler cannot check a spelling error in **printf** or **scanf**.

The Basics of a C Program

- Constants and Variables
- Expressions
- Statements
- Statement Blocks
- Function Blocks

Constants and Variables

A **constant** is a value that never changes.

A **variable** can be used to present different values.

For instance:

```
        i = 1;
```

Symbol **1** is a **constant** because it always has the same value (**1**).
Symbol **i** is assigned the **constant 1**.
i contains the value of **1** after the statement is executed.

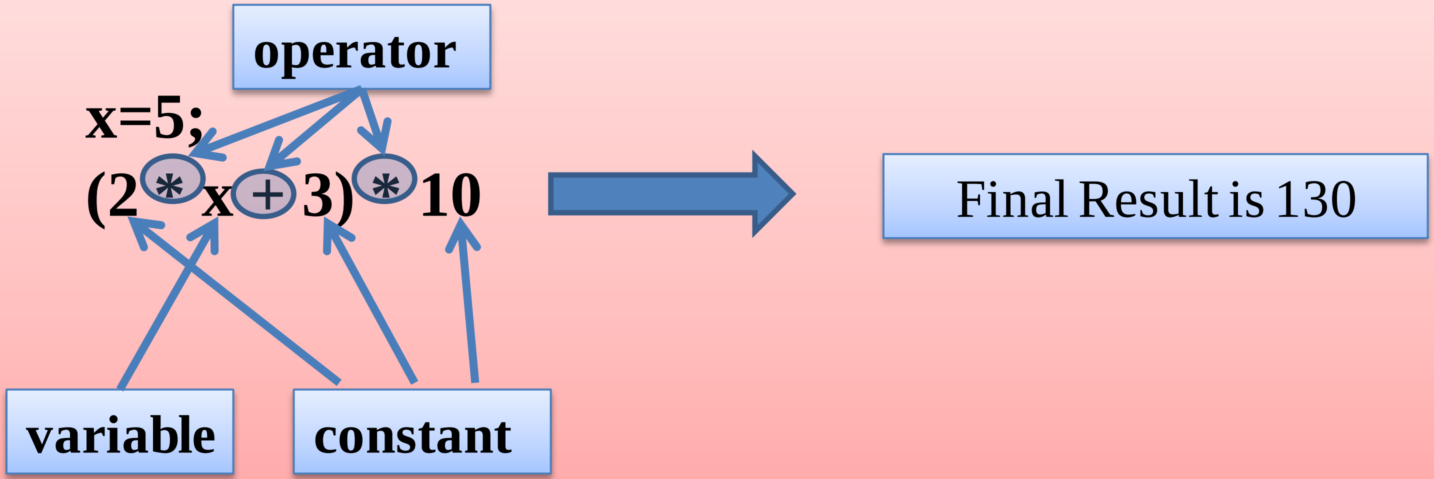
```
        i = 10;
```

After it is executed, **i** is assigned the value of **10.** Value of **i** is changable and **i** is called a **variable.**

Expressions

An **expression** is a combination of constants, variables and operators.

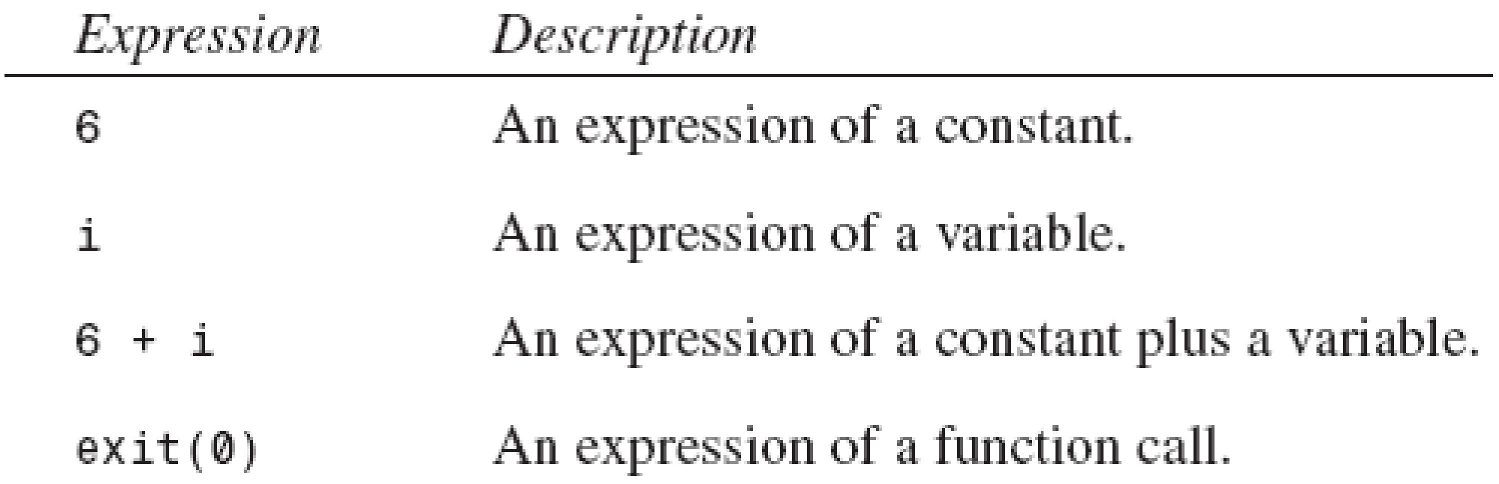
For instance:



Expressions

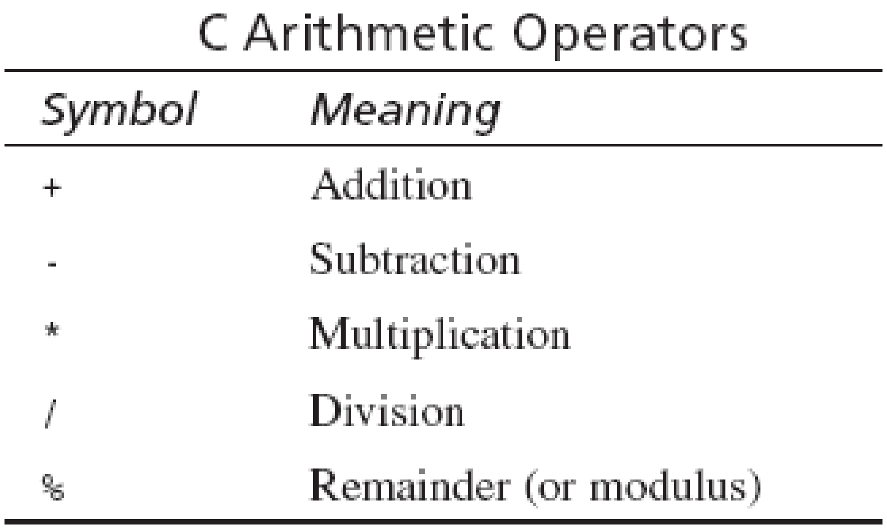
10 * (4 + 5) → Final Result is 90

80 / 4 → Final Result is 20



Expression	*Description*
`6`	An expression of a constant.
`i`	An expression of a variable.
`6 + i`	An expression of a constant plus a variable.
`exit(0)`	An expression of a function call.

Operators



C Arithmetic Operators

Symbol	*Meaning*
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Remainder (or modulus)

% is used to obtain remainder of the first operand divided by the second operand and is also called **modulus** operator.

For instance:

Operand 1 and 2

6 % 4

Final Result is 2

Because 1*4 + 2 = 6

remainder

Operators

Precedence property:

Multiplication, division and remainder operators have a higher precedence than the **addition and substraction** operators.

For instance:

2 + 3 * 10 → Final Result is 32 not 50

Because 3 * 10 is calculated first and then 2 is added into the multiplication result .

If you want to increase precedence of the addition and the substraction operators, you should use parentheses.

(2 + 3) * 10 → Final Result is 50

Identifiers

Identifiers in C:

❖ Function names,

❖ Variable names

❖ Reserved keywords

The set of characters for the valid identifier:

❖ Characters **A** through **Z** and **a** through **z**

❖ Digit characters **0** through **9** (not as first character of identifier)

❖ The underscore character (_)

For instance **: stop_sign** , **Loop3** and **_pause** are valid identifiers.

Illegal characters :

❖ (+, -, *, /) and dot character (.)

❖ Apostrophes (‘) or quotes (“)

❖ Any other special symbols such as *, @, #, ?, and so on.

For instance **: 4flags** , **sum-result**, **method*4**, and **what_size?**

Statements

A **statement** is a complete instruction, ending with a **semicolon (;)**

In many cases, an **expression** can be turned into a **statement** by adding **semicolon** at the end of the **expression**.

For instance:

```
i = 1;
```

is a **statement** consisting of an **expression** **i = 1** and a **semicolon** (;).

```
i = (2 + 3) * 10;
j = 6 % 4;
return 0;
printf(“Hello world …\n”);
```

Statement Blocks

A group of **statements** can form a **statement block** that starts with an **opening brace** (**{**) and ends with a **closing brace** (**}**).

For instance:

```
for ( . . . ) {
        s3 = s1 + s2;
        mul = s3 * c;
        remainder  = sum % c;
}
```

Here **for** is a keyword in C that determines the **statement block.**

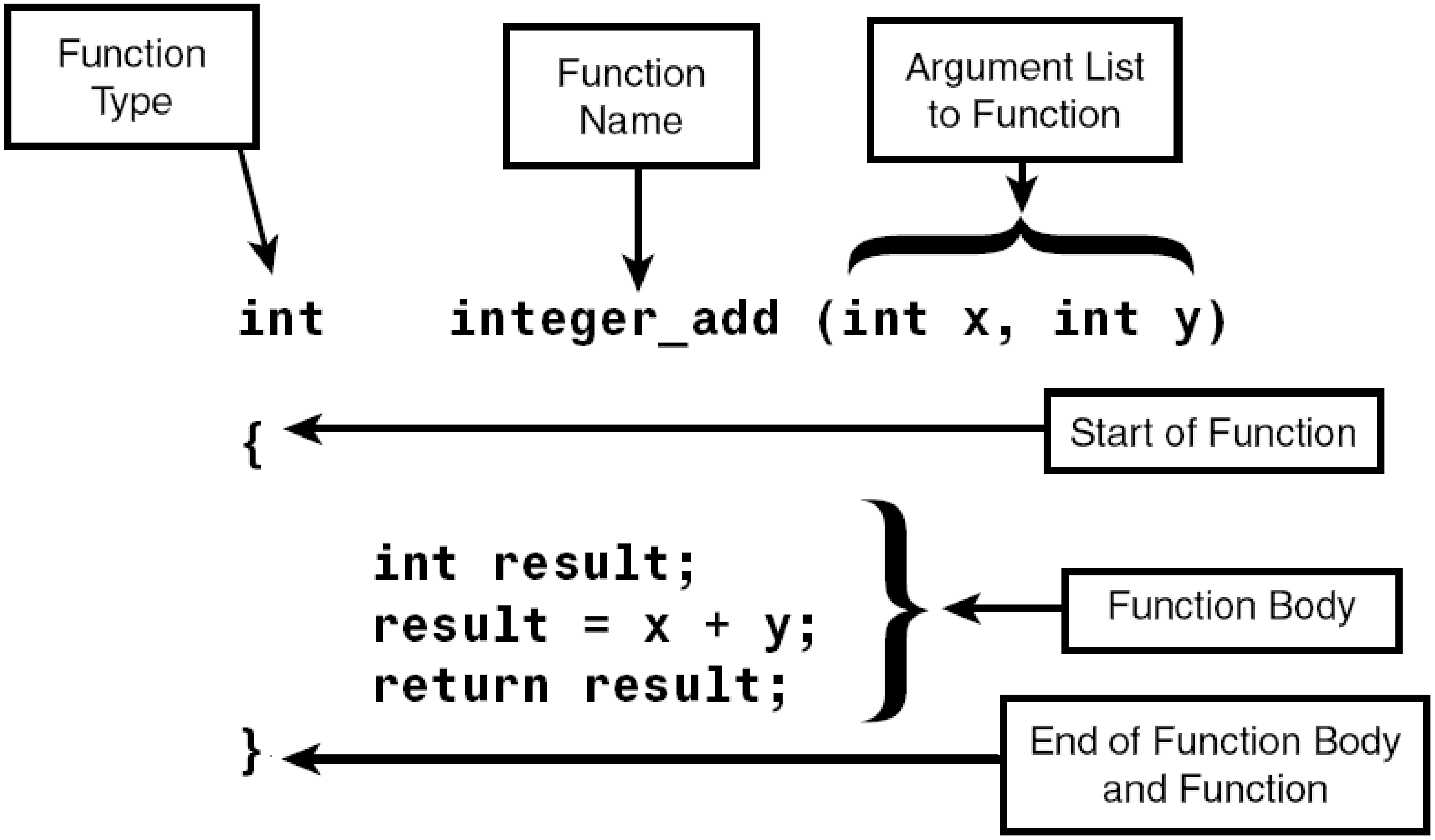
Anatomy of C Function

- Fuctions are the building blocks of C program.
- Main task can be divided into subtask and each subtask can be realized with C functions.
- C is named as structural programming language, because C program is based on one (main function) or more functions.

Main parts of a function:

- The function type
- Function name
- Arguments to the function
- The opening brace
- The function body
- The closing brace

Anatomy of C Function



```
int     integer_add (int x, int y)
{
        int result;
        result = x + y;
        return result;
}
```

Anatomy of C Function

Determining a function's type:

Function type determines the type of return value.

Return value of a function can be used as follows:

```
int  a = func( );

a = func( ) + 7;
```

Giving a Function a Valid Name:

A function name is an **identifier**, it must follow the rules for creating valid **identifiers**.

Function name reflects what the function can do.

For instance: printf function print data on the screen.

Anatomy of C Function

Passing Arguments to C Functions:

Pieces of information passed to functions are known as **arguments**

If no information needs to be passed to a function, leave the argument field blank.

For instance : main ( )

The Function Body:

The function body contains **variable declerations** and other C **statements**.

***: Any **variable declerations** must be placed at the beginning of the function body. Otherwise you get an error message from compiler.