The Essentials of C Programs

BIL104E: Introduction to Scientific and Engineering Computing

Lecture 2

- The Essentials of C Program
- Data Types and Names in C

Basics of the C Program

- Before we go into detail of C programming Language Syntax let us look at the basic definitions:
  - Constants and variables
  - Expressions
  - Statements
  - Statement blocks
  - C function types and names
  - Arguments to functions
  - The body of a function
  - Function calls

Basics of the C Program

Basic elements : expressions, statements, statement blocks, and function blocks.

But first, we need to learn two smaller but important elements: **constant** and **variable**, which make up expressions.

- **Constants and Variables :**
  - As its name implies, a constant is a value that never changes. A variable, on the other hand, can be used to present different values.

Basics of the C Program

- Variables are value containers
- Compiler associates with a variable a memory location
- Value of a variable at any time is the value stored in the associated memory location at that time

Basics of the C Program

- Variable names correspond to locations in the computer's memory
- Every variable has a name, a type, a size and a value
- Whenever a new value is placed into a variable (through **`scanf`**, for example), it replaces (and destroys) the previous value
- Reading variables from memory does not change them

Basics of the C Program

Variables and Constans

C provides the programmer with FOUR basic data types:

- **INTEGER**
- **CHAR**
- **FLOAT**
- **DOUBLE**

Basics of the C Program

INTEGERS

These are whole numbers, both *positive* and *negative*. *Unsigned integers* (positive values only) are supported. In addition, there are short and long integers.

The keyword : **`int`**

```
int letter;
letter = 20;
```

Basics of the C Program

CHARS

These are whole numbers, both *positive* and *negative*. *Unsigned integers* (positive values only) are supported. In addition, there are short and long integers.

The keyword : **`char`**

```
char sum;
sum = 'A';
```

Basics of the C Program

FLOAT

There are numbers which contain fractional parts, both positive and negative.

The keyword : **`float`**

```
float money;
money = 125.50;
```

Basics of the C Program

DOUBLE

There are numbers which contain fractional parts, both positive and negative.

The keyword : **`double`**

```
double big;
big = 125E+5;
```

Basics of the C Program

The keyword TYPEDEF

The typedef keyword is used to create a new name for an existing data type. In effect, typedef creates a synonym. For example, the statement

```
typedef int tamsayi;
tamsayi count;
```

Basics of the C Program

- 10293845 is an integer constant
- 12.3456 is a real constant
- "What a nice day!" is a character constant

Basics of the C Program

Variable Type	Keyword	Bytes Required	Range
Character	char	1	-128 to 127
Integer	int	2	-32768 to 32767
Short integer	short	2	-32768 to 32767
Long integer	long	4	-2,147,483,648 to 2,147,438,647
Unsigned character	unsigned char	1	0 to 255
Unsigned integer	unsigned int	2	0 to 65535
Unsigned short integer	unsigned short	2	0 to 65535
Unsigned long integer	unsigned long	4	0 to 4,294,967,295
Single-precision floating-point	float	4	1.2E-38 to 3.4E38[1]
Double-precision floating-point	double	8	2.2E-308 to 1.8E308[2]
[1]Approximate range; precision = 7 digits.			
[2]Approximate range; precision = 19 digits.			

Basics of the C Program

The Escape Character (\)

In the C language, the backslash (\) is called the escape character; it tells the computer that a special character follows. For instance, when the computer sees \ in the newline character \n, it knows that the next character, n, causes a sequence of a carriage return and a line feed. Besides the newline character, several other special characters exist in the C language, such as the following:

Character **Description** :

\b	The backspace character; moves the cursor to the left one character.
\f	The form-feed character; goes to the top of a new page.
\r	The return character; returns to the beginning of the current line.
\t	The tab character; advances to the next tab stop.

Basics of the C Program

Variable Initialisation

- All variables are initially undefined
- Initialisation in the declarations

Examples:

```
double W=1.2, z=5.678, mass=4.56;
int year=1998, count=0;
const months= 12;
```

Basics of the C Program

```
#include < stdio.h >
main()
{
    int     sum;
    float money;
    char  letter;
    double pi;

    sum = 10;          /* assign integer value */
    money = 2.21;      /* assign float value */
    letter = 'A';      /* assign character value */
    pi = 2.01E6;       /* assign a double value */

    printf("value of sum = %d\n", sum );
    printf("value of money = %f\n", money );
    printf("value of letter = %c\n", letter );
    printf("value of pi = %e\n", pi );
}
```

```
Sample output:
value of sum = 10
value of money = 2.210000
value of letter = A
value of pi = 2.010000e+06
```

Basics of the C Program

Be careful not to initialize a variable with a value outside the allowed range. Here are two examples of out-of-range initializations:

```
int weight = 1000000;
unsigned int value = -2500;
```

The C compiler doesn't catch such errors. Your program might compile and link, but you might get unexpected results when the program is run.

Basics of the C Program

Variable name examples

```
current
decay_rate
pressure
an_identifier_with_a_long_name
the_best_program
```

Basics of the C Program

- **Expressions**

  – An expression is a **combination of constants, variables, and operators** that are used to denote computations. For instance, the following:

    - `(2 + 3) * 10` is an expression that adds `2` and `3` first, and then multiplies the result of the addition by `10`. (The final result of the expression is `50`.)
    - Similarly, the expression `10 * (4 + 5)` yields `90`. The `80/4` expression results in `20`.

Basics of the C Program

- **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.

Basics of the C Program

- **Complex expressions:**

- `1.25 / 8 + 5 * rate + rate * rate / cost`
- `Pow(x,3)+pow(y,3)-5`

Basics of the C Program

- **Statements**

  – In the C language, a statement is a complete instruction, ending with a semicolon. In many cases, you can turn an expression into a statement by simply adding a semicolon at the end of the expression.

  – For instance, the following

  i = 1; is a statement.

  You may have already figured out that the statement consists of an

  expression of i = 1 and a semicolon (;).

Basics of the C Program

Here are some other examples of statements:

```
i = (2 + 3) * 10;
i = 2 + 3 * 10;
j = 6 % 4;
k = i + j;
k = pow(x,3);
```

Also, below are C statements.

```
return 0;
exit(0);
printf ("This is my first C program.\n");
```

Basics of the C Program

- **Statement Blocks**

  – A group of statements can form a statement block that starts with an opening brace ({) and ends with a closing brace (}). A statement block is treated as a single statement by the C compiler.

  – For instance, the following

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

  is a statement block that starts with { and ends with }

Basics of the C Program- Operators

- An *operator* is a symbol that instructs C to perform some operation, or action, on one or more operands. An *operand* is something that an operator acts on. In C, all operands are expressions. C operators fall into several categories:

  - Mathematical operators
  - The assignment operator
  - Relational operators
  - Logical operators

Basics of the C Program

- **Mathematical Operators**

As you've seen, an expression can contain symbols such as +, -, *, and /. In the C language, these symbols are called arithmetic operators.

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

Basics of the C Program

- **Mathematical Operators**

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	`x / y`
Modulus	%	$r \bmod s$	`r % s`

Basics of the C Program

- You may already be familiar with all the arithmetic operators, except the remainder (%) operator. % is used to obtain the remainder of the first operand divided by the second operand. For instance, the expression

  6 % 4 yields a value of 2 because 4 goes into 6 once with a remainder of 2

Basics of the C Program

- Variables and constants can be processed by using operations and functions appropriated to their types.

Basics of the C Program

+	Addition, unary plus
-	Subtraction, unary minus
*	Multiplication
/	Division
%	Remainder

exp()	Exponentiation (MATH.H)
pow()	Power (MATH.H)

Basics of the C Program

Priority Rules:

- All powers & exponentiations are performed first; consecutive power & exponentiations are performed from right to left
- All multiplications and divisions are performed next; in the order in which they appear from left to right
- Additions and subtractions are performed last, in the order in which they appear from left to right

Basics of the C Program - Operations

Examples:

To calculate b^2-4ab ➔ `pow(b,2)+4*a*c`

➔ `10/2*pow(pow(2,3), 3) = 2560`
➔ `10/2 * pow(8,3)`
➔ `10/2 * 512`
➔ `5 * 512 = 2560`

➔ `10/5*2 = 2 * 2 = 4`

➔ `10/(5*2) = 10 / 10 = 1`

Basics of the C Program - Operations

Assignment (Assignment Statement)

- Form:
```
variable = expression;
```

- Assigns the value of **expression** to **variable**
- Assignment is not a statement of algebraic equality; it is a replacement statement
- Examples
```
Density = 2000.0;
Volume = 3.2;
Mass = Density*Volume;
WeightRatio = log(Mass/90.);
```

A Simple Program- Adding two integers

```
1
2   /* Addition program */
3  #include <stdio.h>
4
5  int main()
6  {
7     int integer1, integer2, sum;        /* declaration */
8
9     printf( "Enter first integer\n" );  /* prompt */
10    scanf( "%d", &integer1 );           /* read an integer */
11    printf( "Enter second integer\n" ); /* prompt */
12    scanf( "%d", &integer2 );           /* read an integer */
13    sum = integer1 + integer2;          /* assignment of sum */
14    printf( "Sum is %d\n", sum );       /* print sum */
15
16    return 0;  /* indicate that program ended successfully */
17 }
```

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

Initilaze Variables

Input

Sum

Print

Output

Basics of the C Program-Functions

- **Anatomy of a C Function**

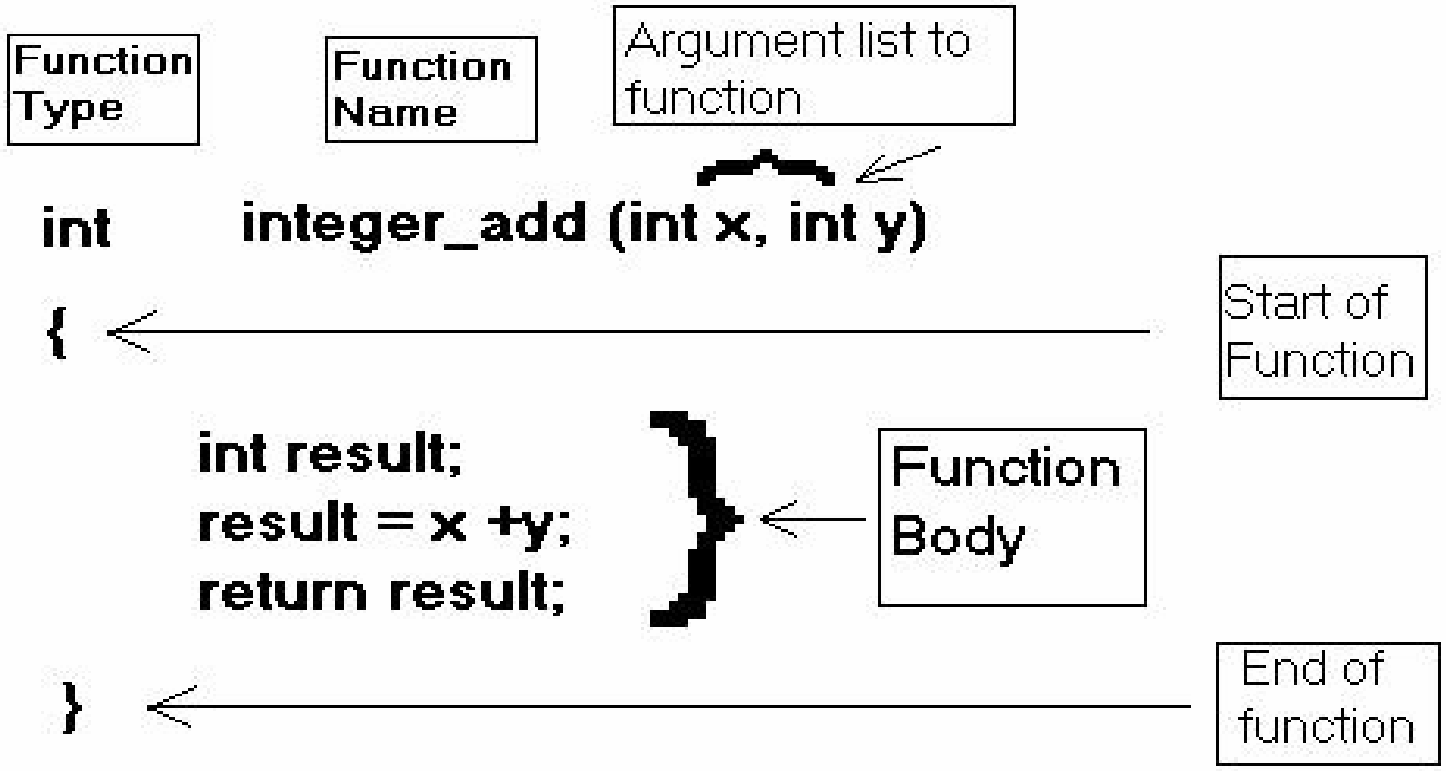
  Functions are the building blocks of C programs.

  Besides the standard C library functions, you can also use some other functions made by you or another programmer in your C program.

  In Lecture 1 you saw the `main()` function, as well as two C library functions, `printf()` and `exit()`.

  Now, let's have a closer look at functions....

Basics of the C Program



Basics of the C Program

- **Giving a Function a Valid Name**
  - A function name is given in such a way that it reflects what the function can do. For instance, the name of the printf() function means "print formatted data."
  - There are certain rules you have to follow to make a valid function name. The following are examples of illegal function names in C:

Illegal Name	**The Rule**
2 (digit)	A function name cannot start with a digit.
* (Asterisk)	A function name cannot start with an asterisk.
+ (Addition)	A function name cannot start with one of the arithmetic signs that are reserved C keywords.
. (dot)	A function name cannot start with ..
total-number	A function name cannot contain a minus sign.
account'97	A function name cannot contain an apostrophe.

Basics of the C Program

- *A function is named*. Each function has a unique name. By using that name in another part of the program, you can execute the statements contained in the function. This is known as *calling* the function. A function can be called from within another function.

- *A function is independent*. A function can perform its task without interference from or interfering with other parts of the program.

- *A function performs a specific task*. This is the easy part of the definition. A task is a discrete job that your program must perform as part of its overall operation, such as sending a line of text to a printer, sorting an array into numerical order, or calculating a cube root.

- *A function can return a value to the calling program*. When your program calls a function, the statements it contains are executed. If you want them to, these statements can pass information back to the calling program.

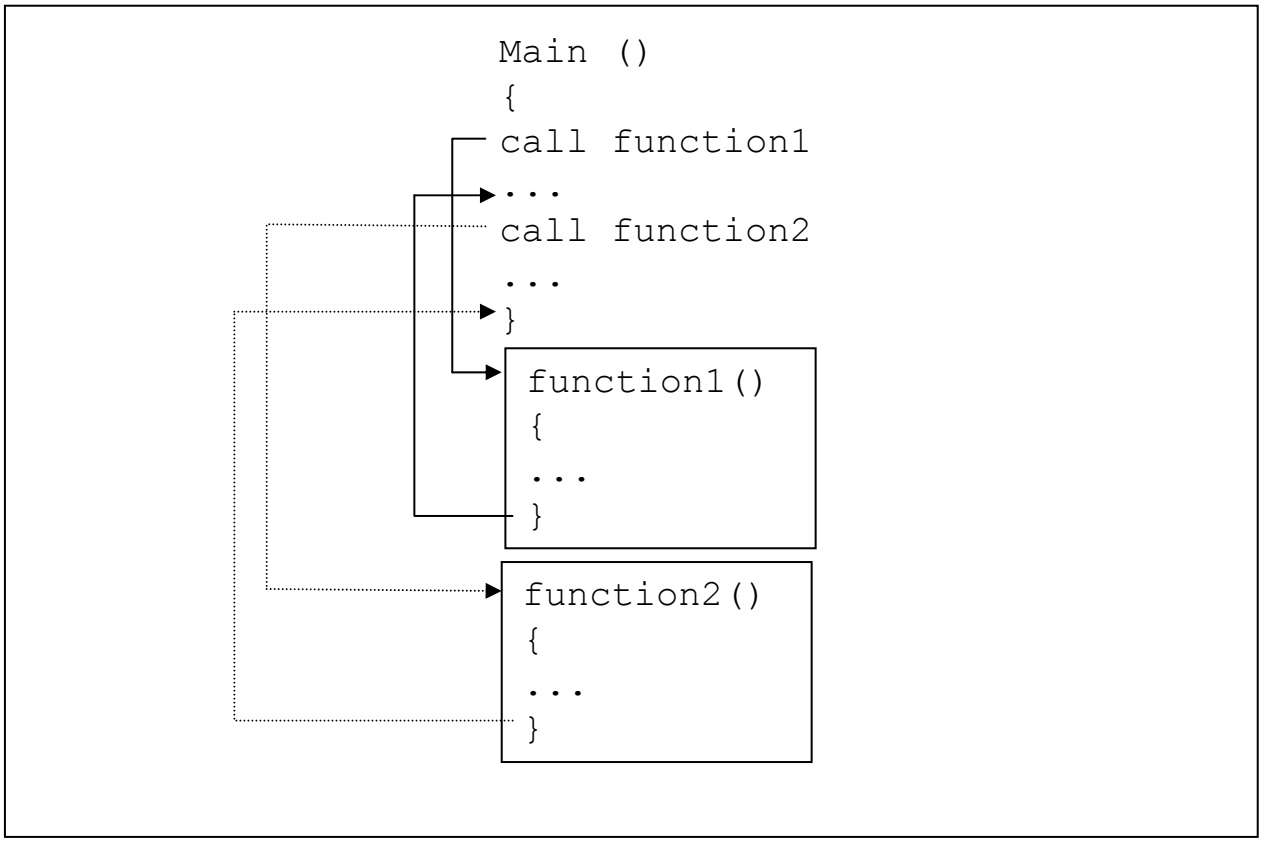
Basics of the C Program

```c
/* This function adds two integers and returns the result */

#include <stdio.h>

int integer_add( int x, int y );


int main(){
   int sum;

   sum = integer_add( 5, 12);       /* Function Call */
   printf("The addition of 5 and 12 is %d\n", sum);
   return 0;
}

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

Basics of the C Program

How a function works

A C program does not execute the statements in a function until the function is called by another part of the program.



Basics of the C Program

Passing arguments to a function.

The number of arguments and the type of each argument must match the parameters in the function header and prototype.