Lecture 4.1

Topics

1. Constants in C – Revisited
2. Variables – Introduction
3. Graphical Representation of Structured Design/Solution – Flow Chart
4. Formatted Input – `scanf()`

---

1. Constants in C – Revisited

Recall that constants in C can be integer, floating-point, character, or string.

In C, there are three different ways to code constants:

a. Literal constants
b. Defined constant
c. Memory constants

Example 1

```
/**
 *Program Name:  cis26L0411.c
 *Written By:    T. T. Nguyen
 *Discussion:    Constants
 */
#include <stdio.h>
#define PI 3.14
const int const_PI = 3.14;
const int i_My_Constant = 100;

int main( void ) {
  printf( "Value of a literal constant : %d\n", 100 );
  printf( "Value of a defined constant : %f\n", PI );
  printf( "Value of a memory constant : %d\n", i_My_Constant );

  return 0;
}
```

```
OUTPUT
Value of a literal constant : 100
Value of a defined constant : 3.140000
Value of a memory constant : 100
```

2. Variables – Introduction

In C, every data value must also have and/or inherit a type.

Consider the following statements,

- One should not just say "**5 is a number**" because it is an incomplete specification in C.
- Even saying "**5 is an integer value**" is somewhat incomplete.
- A complete statement may be as "**5 is an integer of type `int`**" or "**5 is an `int`**".

Data type is a mechanism that C enforces how a value should be stored. One then retrieves and uses the value depending on need (or required operations). The process of storing and retrieving values will most likely involve **variable**.

So then, what is a variable?

2.1 Definition

In C, a variable is a name that the user can declare (specify) and then use to store data of a given type.

A variable is used to store data and, thus, one should be able to identify its storage and value.

A variable is also called an "**identifier**". An identifier allows the system (and programmer) to identify "things" such as, of course, variables, functions, user's defined types, etc.

User can choose any name for the variable. A variable (name) can be made up with combination of

- Alphabets,
- Digits, and
- In some cases, with the underscore ('_') and/or dollar sign ('**$**').

In most cases, the use of underscore and dollar sign is not needed or recommended and should be avoided.

2.2 Naming a Variable

There are some general guidelines/suggestions for making up a variable name as follows,

(1) A variable name should start with a **lowercase alphabet**, and

(2) A variable name can have several words (that means multi-word name). Each word should start with an **uppercase alphabet** except the first word (i.e., after the first word, each following word should start with an uppercase character); and

(3) Digits can be used in combination with alphabets (of course, digits cannot be the first characters for variable names).

Note that a variable must be declared before it can be used.

2.3 Examples

```
int iCounter;    // A declaration or declaring statement
                 // Declaring iCounter to be a variable of
                 // type int, or
                           // Declaring iCounter to be an int
long lSum;               // Declaring lSum to be a long

float fVar1;       // Declaring fVar1 to be a float

double dVar2;    // Declaring dVar2 to be a double

char cValue1;      // Declaring cValue1 to be a char

iCounter = 0;      // Using iCounter – assigning 0 to iCounter

dVar2 = iCounter + 4.5; // Take iCounter added to 4.5 and
                        // assign to dVar2
```

Of course, variables are used in the outputting (such as in `printf()` statements) or inputting (such as `scanf()` statements to be discussed later).

In a C program, all variables must be declared first (at the top of main() function) before any expressions and operations involving these variables are allowed.

3. Graphical Representation of Structured Design/Solution – Flow Chart

Recall that computer programming is a technique that uses computer programming language to manipulate data in order to produce the desired output. That means one must write code, where code should be organized based on how the solution logic might be.

Most of the time, one must design the solution before writing code. The **design step** represents **a possible** solution to the problem. There can be different designs that would also produce the required solution.

There are several **representations** of a design such as

- Logical and detailed description,
- Flowchart,
- Pseudo-code, etc.

Let's consider the flowchart and some simple and basic components.

3.1 Flowchart

A program must have a ***start***. This is shown as **start block**. From this starting point, the execution (or execution flow or just flow) will begin. A program must also end after a successful run (or execution). This is shown with a ***stop*** or ***end*** block.

To most problems, one must have some data to work with. This is called input data or just **input block**. Thus, getting or preparing input is basically the first operational step in the flow.

There will be several steps or required operation(s) to produce the results at the end. These are called **processing blocks**. These operational or processing blocks are problem-specific that may perform tasks on given input (also called output of a processing block).

To the end, the produced results are called the program output. Normally, the output may be displayed on screen and/or stored in file.

A **flowchart** uses some graphical components to depict the above steps and components. A few of the flowchart components are depicted in **Figure 1** below.

3.2 Flowchart and Flowchart Components

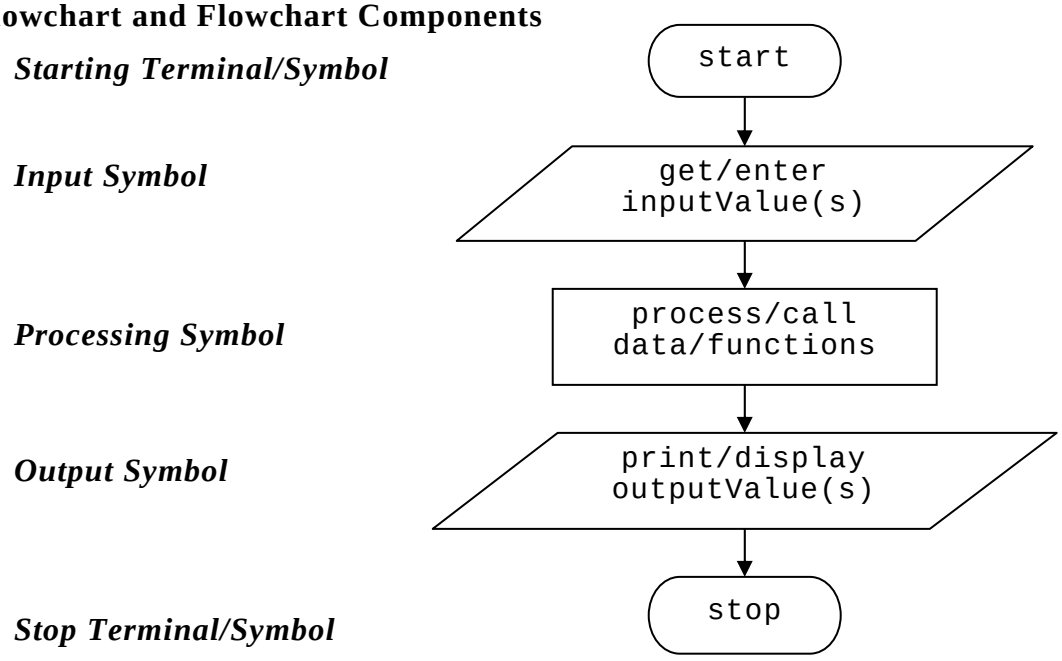

Figure 1 Simple flowchart and components

Elaboration and explanation on the above chart will be given in class.

4. Formatted Input – `scanf()`

In C, the standard formatted input function is `scanf()` (*scan formatted*). The general syntax is given below.

```
scanf( "format string", address list );
```

The function `scanf()` has two parts:

(1) A format string, and
(2) An address list.

4.1 The Format String

The format string for `scanf()`

- Is enclosed in double quotes, and
- Contains one or more field specifications.

The set of characters in the format string shows the exact pattern for the data to be input. Thus, it is

- Not recommended to include delimiters (such as commas, colons, semicolons, etc.) in the format string.
- Recommended to have only one input for each `scanf()` call.

The field specification begins with a percent sign (**%**) and the conversion string. It is given as follows,

```
%<flag><maximum_width><size>conversion_code
```

The components in the conversion code are about the same as those given in the `printf()` field specification with a few exceptions and limitations given below.

(a) With the exception of the character format (**c**), `scanf()` will skip leading white space (e.g., spaces, tabs, and newlines).
(b) If the conversion code is **c** (for character), then `scanf()` will read exactly one character, including the whitespace characters. The data (string) being entered must follow exactly the description of the format string. To skip leading whitespace when reading character data, include a space before the field specification as follows,

```
" %c"
```

For numeric data, `scanf()` will

- Read until it encounters the trailing whitespace character, or
- If the width specifier was given, read until maximum number of characters have been processed, or
- Stop reading if there is a whitespace character if found before the maximum number of characters is reached.

In any reading operations, `scanf()` will stop when

- There is no more input by keying **end-of-file** (EOF). This **EOF** is simulated by `<ctrl+z>` for PC or `<ctrl+d>` in UNIX, or
- There is an invalid character encountered when it is trying to convert the input to the stored data type. For examples, nonnumeric character when trying to read a number (valid numeric characters are +, -, digits, and one decimal point).

4.2 The Address List

The address list should have the same number of addresses as the field specifications. An address is specified with an ampersand (‘**&**’) in front of a variable name. In C, the ampersand (‘**&**’) is known as the **address operator**. For examples,

```
int iX;
scanf( "%d", &iX );
```

Conversion Specification (%conversion)	**Input Result**
%c	Character
%s	Character string Begin with the first non-whitespace character and include everything up to the next whitespace character
%d	Signed decimal integer
%f	Floating-point number

Conversion Modifier	**Meaning**
digit	Maximum field width Input stops when the maximum field width is reached or when the first whitespace character is encounter
h	%hd and %hi ---> short int
l	%ld and %li ---> long int %le and %lf ---> double
L	%Le and %Lf ---> long double

`scanf()` function returns the number of items that it successfully read. If it read no items then `scanf()` returns the value of **`0`**. It returns **`EOF`** when it encounters end of file condition.

Never end the format string with whitespace. If there is a whitespace at the last location of the format string, the system may hang up when running the program.

```
int iVar;
scanf( "%d", &iVar );
```