Lecture 05: C Fundamentals

Md. Taslim Arefin
Associate Professor, Dept. of ETE
Daffodil International University

The C Character Set

- Set of characters that are used as building blocks to form basic program elements
- The C character set consists of
  - The 52 upper- and lower-case letters of the Latin alphabet
  - The ten decimal digits
  - Certain special characters

```
+    -    *    /    =    %       &      #
!    ?    ^    "    '    ~       \      |
<    >    (    )    [    ]       {      }
:    ;    .    ,    _    (blank space)
```

Special characters used in C

Identifiers (1/2)

- Names given to various program elements such as variables, functions, labels, and other user defined items
- Naming rule for identifiers:
  - Can be a combination of letters, digits and underscore (_), in any order
  - The first character of an identifier must not be a digit

Correct	Incorrect
`x`	`"x"`
`y12`	`12y`
`nepal`	`nepal's`
`item_1`	`item 1`
`_temp`	`4th`
`order_no`	`order-no`

Identifiers (2/2)

- In an identifier, upper- and lowercase are treated as different
  - For e.g., the identifier **`count`** is not equivalent to **`Count`** or **`COUNT`**
- There is no restriction on the length of an identifier
- However, only the first 31 characters are generally significant
  - For e.g., if your C compiler recognizes only the first 3 characters, the identifiers **`pay`** and **`payment`** are same

Keywords

- Keywords are reserved words that have standard, predefined meanings in C
- You cannot use a keyword for any other purpose other than as a keyword in a C program
  - For e.g., you cannot use a keyword for a variable name
- The ANSI C defines 32 keywords

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

Data Types

- Data types determine the way a computer organizes data in memory
- Determines how much space it occupies in storage and how the bit pattern stored is interpreted
- Types can be either predefined or derived
- The predefined types in C are the basic types and the type void
- Basic types consist of the integer types and the floating types

Basic Data Types

- The C language supports four basic data types, each of which are represented differently within the computer memory

Data Type	Description	Typical Memory Requirements	Minimal Range
`char`	A single character	1 byte	-128 to 127 or 0 to 255
`int`	An integer value, typically reflecting the natural size of integers on the host machine	2 bytes or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
`float`	Single precision floating-point number	4 bytes	1E-37 to 1E+37 with six digits of precision
`double`	Double-precision floating-point number	8 bytes	1E-37 to 1E+37 with ten digits of precision

Variables

- Named location in memory
- Used to hold a value that can be modified by a program
- Has a type associated with it
- Specifying a variable requires two things:
  - you must give it a name, and
  - you must identify what kind of data you propose

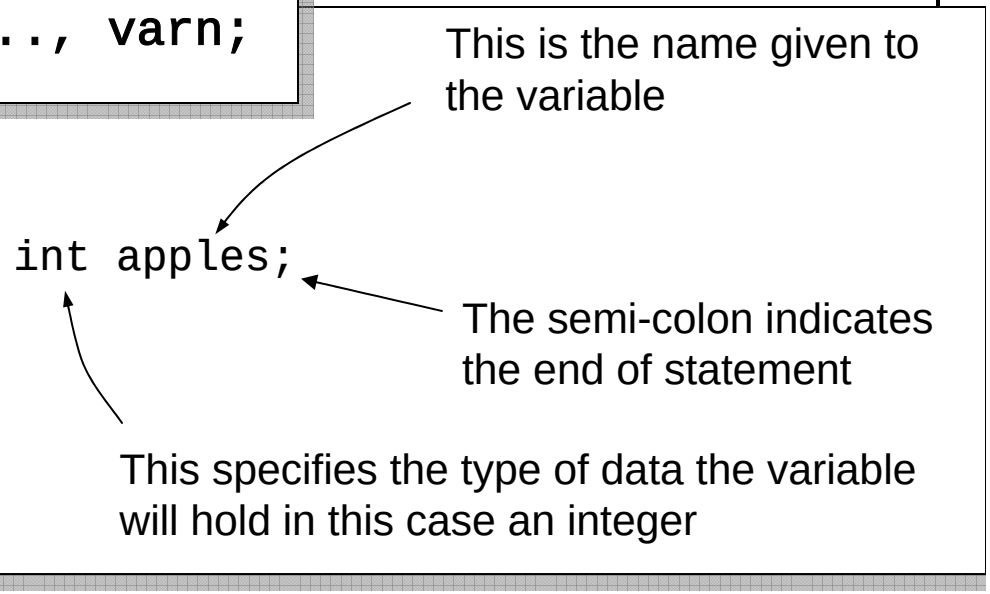
Variable Declaration

- When you declare a variable, you instruct the compiler to set aside storage space for the variable
- All variables must be declared before you can use them

Syntax:

```
data-type var1, var2, var3, ..., varn;
```

```
int count;
int m,n;
char letter;
double profit, loss;
```



Modifying the Basic Data Types

- The basic data types can have various ***qualifiers*** preceding them
- The four qualifiers are **`signed`**, **`unsigned`**, **`long`**, and **`short`**
  - The **`int`** type can be qualified by **`signed`**, **`short`**, **`long`** and **`unsigned`**
  - The **`char`** type can be modified by **`unsigned`** and **`signed`**
  - You can also apply **`long`** to **`double`**
- When a type qualifier is used by itself, then **`int`** is assumed

Modifying the Basic Data Types

- The intent is that **`short`** and **`long`** should provide different lengths of integers where practical; **`int`** will normally be the natural size for a particular machine
  - **`short`**s and **`int`**s are at least 16 bits, **`long`**s are at least 32 bits, and **`short`** is no longer than **`int`**, which is no longer than **`long`**
- **`signed`** and **`unsigned`** values require same storage size but they differ in the way that their high-order bit is interpreted
- The type **`long double`** specifies extended-precision floating point

Data Types Ranges and Memory Requirements

Data Type	Typical Size in Bytes	Minimal Range
`char`	1	–128 to 127 or 0 to 255
`unsigned char`	1	0 to 255
`signed char`	1	–128 to 127
`int, signed int`	2 or 4	–32768 to 32767
`unsigned int`	2 or 4	0 to 65535
`short int, signed short int`	2	–32768 to 32767
`unsigned short int`	2	0 to 65535
`long int, signed long int`	4	–2147483648 to 2147483647
`unsigned long int`	4	0 to 4294967295
`float`	4	1×10+37 to 1×10–37 with six digits of precision
`double`	4	1×10+37 to 1×10–37 with ten digits of precision
`long double`	10	1×10+37 to 1×10–37 with ten digits of precision

Constants

- Like a *variable*, a **constant** is a data storage location used by your program
- Unlike a variable, the value stored in a constant can’t be changed during program execution
- Every constant has a type that is determined by its value and its notation
- C has two types of constants: ***literal constants*** and ***symbolic constants***
- A **literal constant** is a value that is typed directly into the source code wherever it is needed
- Literal constants are also only referred as constants

Numeric Constants

- ***Floating-point constants:***
  - written with a decimal point is a floating-point constant
  - represented by the C compiler as a double-precision number
- ***Integer constants:***
  - A constant written without a decimal point

Floating-point constants		Integer constants
`123.456`	`1.23E2`	`1245 (base 10)`
`0.019`	`4.08e6`	`0147 (base 8)`
`100.`	`0.85e-4`	`0x1F (base 16)`

Character Constants

- A single character, enclosed in apostrophes
- Character constants have integer values that are determined by the computer's particular character set
- Most computers, and virtually all personal computers, make use of the ASCII character set
- In ASCII, each individual character is numerically encoded with its own unique 7-bit combination

Character constants examples

```
'A'    'c'    '#'    ' '    '3'
```

Constant	ASCII Value
'A'	65
'x'	120
'3'	51
'?'	63
' '	32

Escape Sequences

- Certain nonprinting characters, as well as the backslash (\) and the apostrophe ('), can be expressed in terms of ***escape sequences***
- Begins with a backward slash and is followed by one or more special characters
- Represents a single character
- A character constant written in the form of escape sequence is called backslash character constant

Character	**Escape Sequence**	**ASCII Value**
bell (alert)	`\a`	007
horizontal tab	`\t`	009
newline	`\n`	011
quotation mark	`\"`	034
apostrophe (')	`\'`	039
backslash (\ )	`\\`	092

String Constants

- Consists of any number of consecutive characters (including none), enclosed in (double) quotation marks
- A character constant (e.g., 'A') and the corresponding single-character string constant ("A") are not equivalent

```
"green"   "Dhaka, Bangladesh"   "977-01-496567"
"Tk. 1,000"   "The lucky number is: ""2*(I+3)/J"
"      "     "Line 1\nLine 2\nLine 3"   ""
"\tTo continue, press the \"Enter\" key\n"
```

Symbolic Constants

- Constant that is represented by a name (symbol) in your program
- Like a literal constant, a symbolic constant can't change
- Whenever you need the constant's value in your program, you use its name as you would use a variable name
- C has two methods for defining a symbolic constant: the **`#define`** directive and the **`const`** keyword

```
#define PI 3.14159
...
...
area = PI * (radius)*(radius);
```

```
const double PI = 3.14159;
...
...
area = PI * (radius)*(radius);
```

Initialization of Variables

- When a variable, its initial value is undefined
- Before using a variable, you should always initialize it to a known value
- To initialize a variable, the declaration must consist of a data type, followed by a variable name, and equal sign (=) and a literal constant of the appropriate type

```
char ch = 'a';
int first = 0;
float balance = 123.23;
double factor = 0.21023E-6;
unsigned int type = 15U;
long int result = 9786788L;
```

Expressions

- An ***expression*** is any valid combination of different entities for example a constant, a variable, an array element or a reference to a function
- It may also consist of some combination of such entities, interconnected by one or more operators
- ***Simple expressions***
  - The simplest C expression consists of a single item: a simple variable, literal constant or symbolic constant

Expression	**Description**
`PI`	A symbolic constant (defined in the program)
`20`	A literal constant
`rate`	A variable
`-1.25`	Another literal constant

Complex Expressions

- Complex expressions consist of simpler expressions connected by operators
- For example:
  - `2 + 8`
    is an expression consisting of the sub expressions `2` and `8` and the addition operator `+`
- You can also write C expressions of great complexity:
  - `1.25/8 + 5*rate + rate*rate/cost`

Logical Expressions

- Expressions can also represent logical conditions that are either true or false
- In C the conditions true and false are represented by the integer values 1 and 0, respectively

Examples

```
x <= y
x == y
```

Statements

- A ***statement*** is a complete direction instructing the computer to carry out some task
- A statement specifies an action
- There are three different classes of statements in C
  - Expression statements,
  - Compound statements,
  - Control statements

Expression Statements

- An ***expression statement*** consists of an expression followed by a semicolon
- The execution of an expression statement causes the expression to be evaluated

```
a = 3;
c = a+b;
++i;
printf("Area = %f", area);
;
```

Compound Statements

- Consists of several individual statements enclosed within a pair of braces **`{ }`**
- The individual statements may themselves be *expression statements*, *compound statements* or *control statements*
- Provides a capability for embedding statements within other statements
- Unlike an expression statement, a compound statement ***does not*** end with a semicolon

```
{
pi = 3.141593;
circumference = 2. *pi * radius;
area = pi * radius * radius;
}
```

Control Statements

- Used to create special program features, such as logical tests, loops and branches
- Many control statements require that other statements be embedded within them

```
while (count <= n)
{
   print ("x = ");
   scanf ("%f", &x);
   sum += x;
   ++count;
}
```

Statements and White Spaces

- The term **white space** refers to spaces, tabs, and blank lines in your source code
- When the compiler reads a statement in your source code, it looks for the characters in the statement and for the terminating semicolon, but it ignores white space
- An exception: white spaces in string literal constant is not ignored