Introduction To C Programming

Lesson 01

Basic Programming Concepts

Objectives

- Understand general computer concepts
- Understand programming concepts and processes
- Understand rudiments of a C program
- Be able to create, compile and run your first program

Computer Fundamentals

- Computer consists of these parts:
  - input unit
  - output unit
  - memory unit
  - arithmetic logic unit (ALU)
  - central processing unit (CPU)
  - secondary storage unit

Computer Fundamentals

- machine language - pure binary numbers that run the machine
- assembly language - uses symbols and statements that compile into machine language
- high-level languages - more English like statements that are compiled to machine language
  - Advantages
  - Where C fits in
    - Cobol, VB, Fortran, Pascal, C++, scripting languages

Computer Fundamentals

- Assemblers - convert assembly language symbols and statements into machine code
- Compilers - process a language and convert statements into machine code

Origin of C

- Developed in 1972 by Dennis Ritchie and Brian Kernigan of Bell Labs.
- Evolution:
  - Algol 60 (1960)
  - CPL (1963)
  - BCPL (1967)
  - B (1970)
  - C (1972)
  - C++ (1983)

Features of C

- Few restrictions or confining rules
- Rich operator set including bit manipulators
- Useful data types (including pointers and strings)
- Standard run-time library (includes I/O)
- Efficient, Small, Portable
- Modern control structures
- Dynamic memory allocation

Advantages of C

- Efficient
- C is the basis for newer languages such as C++, Java, C#, and JavaScript
- Useful for: user interfaces, communications, control systems, automatic test equipment, operating systems, database managers, computer aided design, spreadsheets, text processors, etc.

Limitations of C

- Flexibility can be a problem for inexperienced programmers
- Easy to write valid C code that generates garbage results
- Minimal run-time checking
- Easy to over-use compact forms of statements

The ANSI C Standard

- K&R C
- ANSI C
  - Defines preprocessor, C language, and standard C library
  - Portability, data type sizes
- This class teaches the ANSI C language
  - Enable ANSI compilation on your home compiler - Read your compiler manual
  - For Visual C++ use /TC and /Za options

Typical C Development Process

- Iterative
  - Edit source file
  - Run preprocessor
    - Expand statements that begin with #
  - Compile
    - Convert to assembler, then to machine language
  - Link your program with standard library
  - Load program into computer memory
  - Execute

Input and Output in C

- stdin, stdout, stderr
- stdio.h
- printf() - print formatted
- scanf() - scan formatted

A First Look At C

```
/* love.c   A first look program   */
#include <stdio.h>
#define COUNT 100
int main()
{
   int i;            /* loop variable */
   printf (“How do I love C?\n”);
   printf (“Let me count the ways!\n”);
   for (i = 0; i < COUNT; ++i)
       printf (“Way number %d\n”, i);

   printf (“I love C at least %d ways.\n”, i);
}
```

A First Look At C (output)

```
How do I love C?
Let me count the ways!
Way number 0
Way number 1
. . .
Way number 99
I love C at least 100 ways.
```

Key Points of a C Program

- Programs are built with functions
- One function must be “main()”
- Variables must be declared before use
- Program execution starts with main()
- Case is significant in C
- C is free format

Hello World

```
/*
**     hello.c
*/

#include <stdio.h>

int main()
{
   printf (“Hello world!\n”);

   return 0;
}
```

Bad Examples

```
/*bad1.c*/int main(){printf(“Hello world!\n”);}
```

Bad Examples

```
/*
b
a
d
.c
*/
int
main
()
{
printf(
“Hello world!\n”
);}
```

Bad Examples

```
/*bad3.c*/
    int main(){
        printf(
            “Hello world!\n”
                            )
                             ;
                              }
```

Tokens of C Programs

- C programs are parsed by the compiler into units, or tokens, of these types:
  - Identifiers
  - Constants
  - Comments
  - Separators
  - Operators
  - Keywords

Identifiers

- First 31 characters are significant
- Must begin with an alphabetic character or an underscore
- Digits are allowable after the first character
- No special characters
- Case is significant

Constants

- There are two types: literal constants and symbolic constants
- Literal Constants:
  - decimal, hexadecimal and octal integers
  - character constants
  - string constants
  - floating-point constants
- Symbolic constants have been #defined to a value and are usually all uppercase

Comments

- Begin with /*
- End with */
- May extend over multiple lines
- May not be nested
- Do not use //. This is a C++ comment - not valid in C though some compilers don’t complain
- Can use preprocessor to remove chunks of code
  - #if 0
  - #endif

Separators

- The punctuation of the language
- Include characters such as:

, ; ( ) { }

Operators

- The verbs of the language. They specify the action to be performed on the program's data, such as:
  - * (multiplication)
  - + (addition)
  - - (subtraction)
  - / (division)
  - % (modulo)
  - We'll cover more later

Keywords

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

Whitespace

- Characters used to separate tokens
  - Space ‘ ‘
  - Tab ‘\t’
  - Newline ‘\n’
    - A newline moves the position to the beginning of the next line
- Any number (at least one) may be used where whitespace is allowed:

printf()

```
printf(“Hello World!\n”);
```

- Tells the computer to print whatever is between the quote marks (called a ‘string’): “Hello World!\n”
- Is a call to a function called ‘printf’
- Is a statement. All statements end with a semicolon.
- Sends an a newline to the screen (\n)

Escape Characters

Escape char	Description
\n	Newline. Move cursor to beginning of next line.
\t	Tab. Move to next tab stop
\r	CR. Move to beginning of line
\a	Alert. Sound the system bell.
\\	Print a backslash
\”	Print a double quote mark
\xdd	Print a hexadecimal number
\ddd	Print an octal number

Examples

```
#include <stdio.h>
/* print on one line with two print statements */
int main() {
    printf(“Hello “);
    printf(“world!\n”);
    /* same as: printf(“Hello world!\n”);
}
/* print multiple lines with one print statement */
int main() {
    printf(“Welcome\nto\nC!\n”);
}
```

scanf()

```
int value1 value2;
int nFieldsRead
nFieldsRead = scanf("%d %d", &value1, &value2);
```

- Tells the computer to read an int from stdin
  - %d is a code which tells scanf to retrieve an int
  - We'll learn codes for other data types later
- Is a call to a function called 'scanf'
- Is a statement
- You must place the & character in front of the variable name - we'll learn more about this later.
  - & is called the "address-of" operator

Example

```
#include <stdio.h>
/* read a an int from stdin and print it to stdout */
int main() {
    int value;
    printf (“Enter integer number: “);
    /* Note use of scanf return value to check for
    ** input error */
    if (scanf(“%d”, &value) == 1) {
        printf(“You entered %d\n”, value);
    }
    else {
        printf(“The value was not an int\n”);
    }
}
```