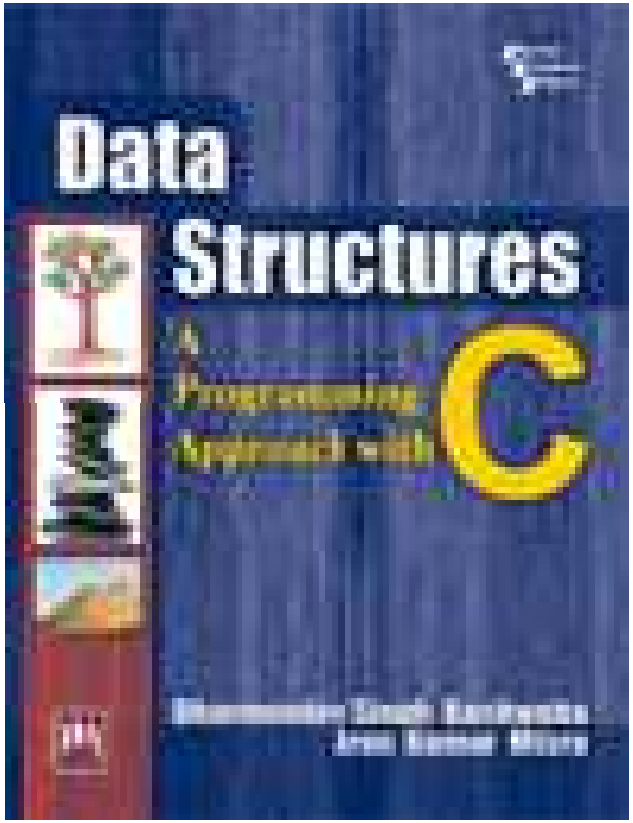

Data Structures: A Programming Approach with C

Chapter 1

A Quick Overview of C Fundamentals

Data Structures: A Programming Approach with C, PHI
Dharmender Singh Kushwaha & A.K.Misra

Computer Program

- A sequence of instructions written to perform a specified task by computer.
- A computer program is written in human-readable, computer programming language called source code.
- Source code is then converted into an executable file by a compiler or executed immediately with the aid of an interpreter.

Compiler

- A computer program or set of programs that transforms source code written in a programming language called the source language into another computer language that is the target language, often in a binary form known as object code or machine code.

Types of Variables

- Global Variables & Local variables
  - Global variable declaration is located outside any of the program's functions.
  - Local variables are declared within the body of a function, and can only be used within that function.

Variable

- A variable is something that stores a value in memory and its value can vary while a program is running.
- Variables have names assigned to them by the programmer.
- Variables can either be used everywhere in the program called global variable or just in one part of the code called local.

Variable

- Global variables can be used throughout the program but their names have to be unique.
- Local variables only exist within a specific section of the program and therefore can have the same name as other local variables in different program sections.

Constant

- Constants are a way of assigning a name of a value. The value of a constant is set and then never changes in a given program.
  - int const max_marks = 100;
  - const int marks = 92;

- C constants can be divided into two major categories:
  - Primary Constants: Eg. int, real, char etc.
  - Secondary Constants: array, pointer, structure etc.

Integers

- The list of available integer types has been expanded to include "signed char" and the following variations:
  - signed char
  - signed int
  - signed long
  - signed short
  - unsigned int
  - unsigned long
  - unsigned short

Flow Control

- There are several types of flow controls in most languages. C supports:
  - if/then/else – used for branching
  - case statements - used for branching
  - for-next loops
  - do while loops
  - repeat until loops

Preprocessor

- The preprocessing is the first step carried out in the C source before it is fed to the compiler.
- The two most common preprocessor directives are #define and #include.

#define

- The #define directive is used to set up symbolic replacements in the source.
- It does textual replacement without understanding. #define statements are used as a crude way of establishing symbolic constants.

Preprocessor

#include

- The "#include" directive brings in text from different files during compilation.
- It just pastes the text from the given file and continues with the compilation.
- The #include directive is used in the .h /.c file.

- #include "dsk.h" // refers to a "user" dsk.h file in the originating directory for the compiler

- #include <dsk.h> // refers to a "system" dsk.h file in the compiler's directory somewhere

A Simple C Program

```
#include <stdio.h>
int main(void)
 {
    printf("hello, world\n");
    return (0);
 }
```

Type Conversion

- Type conversion occurs when an expression has a mixed data types.
- For example: char types are treated as int.
- When types of different size are involved, the result is of the larger size.
- When we add a float and a double, result produced shall be of type double.
- When integer and float types operate, the result is a float.

Type Casting

```
int i = 12;
int  squareroot;
squareroot = sqrt((double)i);
```

- Type casting is made by placing the bracketed name of the required type just before the value, which is double in the example.
- The result of sqrt((double)i); is also a double, but this is automatically converted to an int on assignment to the variable squareroot.

Variable Scope

- The scope of a variable is:
  - The parts of the program that have access to that variable. This is determined by the program section of the declaration of the variable (spatial).
  - The life span of that variable, i.e. the time for which that the variable remains in memory (temporal).

Example

```
int k=4;                /* Global definition */

main()
{
k++;                    /* global variable */
fun()
}

fun()
{
int k=10;               /* Internal declaration */
k++;                    /* Local variable */
}
```

Operators

- An operator is a symbol by which we instruct computer to do a certain mathematical or logical operation. These are:
  - Arithmetic operators
  - Relational Operators
  - Logical Operators
  - Assignment Operators
  - Increments and Decrement Operators
  - Ternary Operators
  - Bitwise Operators
  - Special Operators

Relational Operators

- When we need to compare the relationship between two or more operands, the relational operator come into picture.
- C supports <, <=, >, >=, ==, != relational operators.
- Relational expressions are used in decision making statements.

Logical Operators

- C supports the following logical operators:
  - Logical AND (&&) ,
  - Logical OR (||) &
  - Logical NOT (!) operators.
- These are used to compare or evaluate logical and relational expressions.

Ternary Operator

- The conditional ternary operator consists of 2 symbols - the question mark (?) and the colon (:).
- The syntax for a ternary operator is as follows:

  – exp1 ? exp2 : exp3;

- exp1 is evaluated first.
- If the expression is true then exp2 is evaluated & its value becomes the value of the expression.
- If exp1 is false, exp3 is evaluated and its value becomes the value of the expression.
- We should bear in mind that only one of the expression (exp2 or exp 3) is evaluated.

Bitwise Operators

&	Bitwise AND
|	Bitwise OR
^	Bitwise Exclusive
<<	Shift left
>>	Shift right

printf / scanf

%i or %d	int
%c	char
%f	float
%lf	double (long float)
%s	string

Escape Sequences

\n (newline)

\t (tab)

\v (vertical tab)

\f (new page)

\b (backspace)

\r (carriage return)

\n (newline)

Comments

- Comments are a way of inserting remarks and reminders into a program without affecting its execution and content.
- The compiler treats them as though these are white space or blank characters and these are ignored.

/* ..... comment .....*/ for multiple line comment.

// .....comment..... for single line comment.

Decision Control Instructions

- The **if** statement
- The **if-else** statement
- The conditional operators

switch Statement

- It is a multi way decision statement. It is used in cases where:
  - One variable is tested and all the branches depend on the value of that variable.
  - The variable must be an integral type (int, long, short or char).
  - Each possible value of the variable controls a single branch.

Loops

- C gives us the choice of three types of loop:
  - The while loop keeps repeating an action until an associated test returns false.
  - The do while loops are similar to while loop but the condition is tested after the loop body is executed. This ensures that the loop body is run at least once.
  - *for* loop is used where the loop is executed, a fixed number of times

Functions in C

- Functions make a program modular by separating each independent activity into smaller manageable chunks.
- It's usual to group related functions into libraries and then use a header file so as to be used by others.
- C assumes that every function will return a value.
- If we omit the return type, then ‘int’ is assumed.

Rules of using functions

- If function definition is written after the function call, then it is necessary to declare the function before the function call.
- If function definition is written before the function call statement, then it is not necessary to write function declaration.
- If return type of function is signed int data type, then it not necessary to write function declaration even though function definition is written after the function call.
- Function declaration doesn't reserve any memory space in advance.

The scope and lifetime of variables in functions

- Variables declared within the calling function can't be accessed unless they are passed to the called function as arguments.
- In C, variables can be any one of the four types:
  - Automatic variables
  - External variable
  - Static variable
  - Register variable

Auto Specifier

- The auto specifier indicates that the memory location of a variable is temporary.
- A variable's reserved space in the memory may be erased or relocated when the variable is out of its scope.
- Only variables with block scope can be declared with the auto specifier.

Static Variable

- The static variable is not destroyed on exit from the function; instead its value is preserved, and becomes available again when the function is later called.

- int i; /* This auto variable has block scope and temporary duration */
- static int j; /* This variable has block scope and permanent duration */

The Register Specifier

- It's much faster to access a register than a memory location.
- We apply this specifier to variables when we think it's necessary to put the variables into the computer registers depending upon the kind of application or program.
- register int i; // block scope with the register specifier

The Extern Specifier

- It may be required that a global variable declared in one program file of the application may be required to be used in another program file of the application.
- To help the compiler know this situation, we place a specifier ‘extern’ before the global variable in the program file where it has not earlier been declared as global variable.

```
int x = 0;          // global variable
extern int y;       // global variable y  defined in other program file
int main()
{
        int i;                  // local variable
        /* other program statements */
}
```

Hierarchy of Scopes

- The hierarchy of scopes in decreasing order is:
  - Program Scope
  - File Scope
  - Function Scope
  - Block Scope

Recursion

- Recursion is said for a phenomenon when a function calls itself.
- When a function calls another function and that second function calls the third function then this kind of a function is called nesting of functions.
- But a recursive function is the function that calls itself repeatedly.

Arrays

- Arrays are essentially a way to store many values under the same name.
- An array is a collection of variables of the same type.
- Individual array elements are identified by an integer index.
- In C, the index begins at zero and is always written inside square brackets.
- Eg. int marks[60]; // This declares an array that can store 60 marks

Pointers

- Each memory location in the computer has an address that can be used to store data and we can access this location via the pointer.
- A pointer is a variable that points to or references a memory location where the data is stored.
- The asterisk tells the compiler that we are creating a pointer variable and also specifies the name of the variable.
  - type * variable name;

Address operator

- After declaring a pointer variable, we must point it to something.
- We accompalish this by assigning the address of the variable to the pointer.

```
ptr = &var_name;
```

Pointers

- Pointers are no longer synonymous with the int type. Pointers can only be compared with or assigned to:
  - the integer value 0 that is used to define a null pointer
  - a pointer of the same type
  - a pointer of generic type (a void pointer)
- Any other use of a pointer will generate a warning message upon compilation.

Declaring pointer

```
int *ptr;

float *string;
```

Structures

- C supports a data structure called structure which allows us to club one or more variables with different data types like int, char, float, arrays and other structures together.
- Each variable in a structure is called the structure member.
- To define a structure, we use struct keyword. The syntax of structure definition is as follows:
  - **struct struct_name{ structure_members };**

Declaring Structure

```
struct address
{
   unsigned int house_number;
   char street_name[45];
   int pin_code;
   char country[30];
};
```

Initializing structure members: The dot (.) operator

- To assign values to structure members, we use dot operator (.) between structure name and structure member.

```
address.country = "India";
address.country = "France";
```

Union

- Like structures, unions contain data members whose individual data types can differ from one another.
- However, all the data members that compose a union share the same storage area within the computer's memory where as each members within a structure are assigned its own unique storage area.
- Thus unions are used to conserve memory.

Union

- When a different member is assigned a new value, the new value overwrites the previous members' value.
- Unions may be used in all places where a structure is allowed.
- union book
  {
  int price;
  char publisher_name[50];
  } code;

The Storage Class Qualifiers

- C also provides you with two storage class qualifiers or modifiers for specifying the access of the variables.
  - const &
  - volatile

const

- If we declare a variable with the const quanlifier, the content of the variable will not change after it is initialized throughout the life cycle of the program.
  - Eg.: const double pi = 3.14;

volatile

- This qualifier is mainly used with the problems that are encountered in real-time or embedded systems programming using C.
- The volatile qualifier tells the compiler that the varible is subject to suddenly change its values, for reasons which cannot be predicted from a study of the program itself.
  - Eg. volatile char kbd_ch; // variable to read a char from keyboard

Standard Library Functions

- stdio.h — file input and output
- ctype.h — character tests
- string.h — string operations
- math.h — mathematical functions such as sin() and sqrt()
- stdlib.h — utility functions such as malloc()

Standard Library Functions

- assert.h — the assert() debugging macro
- stdarg.h — support for functions with variable numbers of arguments
- setjmp.h — support for non-local flow control jumps
- signal.h — support for exceptional condition signals
- time.h — date and time