Programming & Data Structures

For

Computer Science & Information Technology

By

THE GATE© ACADEMY

A Forum of IIT / IISc Graduates

www.thegateacademy.com

✆080-40611000

Syllabus for Programming and Data Structures

Programming in C, Recursion, Arrays, Stacks, Queues, Linked Lists, Trees, Binary Search Trees, Binary Heaps, Graphs.

Previous Year GATE Papers and Analysis

GATE Papers with answer key

thegateacademy.com/gate-papers

Subject wise Weightage Analysis

thegateacademy.com/gate-syllabus

Contents

	Chapters	Page No.
#1.	**Introduction to 'C' Programming and Array**	**1 – 51**
	• C Programming	1 – 3
	• Basic Datatypes	3 – 5
	• Variable Types	6 – 10
	• Operators	11 – 14
	• Flow Control Statements	15 – 19
	• Pointers	19 – 28
	• Functions and Variables	28 – 30
	• Strings	30 – 32
	• Working with Files	32 – 37
	• Macro	38
	• Dynamic Memory Allocation	38 – 39
	• Arrays	40 – 41
	• Memory as an Array	41 – 44
	• User Defined Data types	45 – 47
	• Abstract Data Types (ADT)	48 – 49
	• Recursion	50 – 51
#2.	**Linked List**	**52 – 70**
	• Linked List	52 – 56
	• Memory Allocation: Garbage Collection	56 – 57
	• Operation on Linked List	57 – 64
	• Header Linked Lists	64 – 66
	• Circular Linked List	67
	• Doubly Linked List	67
	• Operation on DLL	68 – 70
	• Application of Linked List	70
#3.	**Stacks and Queues**	**71 – 91**
	• Stacks	71 – 72
	• Stack ADT Implementations	72 – 74
	• The Stack Permutation	74 – 78
	• Binary Expression Tree	78 – 82
	• Queue	82 – 84
	• Linked Representation of Queue	84 – 88
	• Queue Operations	88 – 89
	• Queue Applications	89 – 91
#4.	**Trees**	**92 – 111**
	• Binary Tree	92 – 94

• Height Analysis	94 – 95
• Tree Traversals	95 – 99
• Binary Search Trees	100
• Basic BST Operations	100 – 105
• AVL Trees	105 – 106
• Advantage of AVL Tree	106 – 111
#5. Priority Queues (Heaps)	**112 – 123**
• Introduction	112
• Binary Heap	112 – 115
• Array Representation of Binary Heap	115 – 116
• Min-Heap vs. Max-Heap	116
• Basic Heap Operation	116 – 123
#6. Graph	**124 – 132**
• Important Points	124 – 125
• Representation of Graphs	125
• Graph Traversal Algorithm	125 – 129
• Disjoint Set Data Structure	130 – 132
• Connected Components	132
Reference Books	**133**

"Obstacles are those frightful things you can see when you take your eyes off your goal."
...Henry Ford

CHAPTER 1

Introduction to 'C' Programming and Array

Learning Objectives

After reading this chapter, you will know:

1. C Programming
2. Basic Datatypes
3. Variable Types
4. Operators
5. Flow Control Statements
6. Pointing to Data
7. Functions and Variables
8. Strings
9. Working with Files
10. Macro Caveats
11. Dynamic Memory Allocation
12. Arrays
13. Memory as an Array
14. User Defined Data types
15. Abstract Data Types (ADT)
16. Recursion

C Programming

Basic Introduction

C is a general-purpose high level language that was originally developed by Dennis Ritchie for the Unix operating system. The Unix operating system and all Unix applications are written in the C language. C has now become a widely used professional language for various reasons.

- Easy to learn
- Structured language
- It produces efficient programs
- It can handle low-level activities
- It can be compiled on a variety of computers

Facts about C

- C was invented to write an operating system called UNIX
- C is a successor of B language which was introduced around 1970
- The language was formalized in 1988 by the American National Standard Institute (ANSI)
- By 1973 UNIX OS almost totally written in C
- Today C is the most widely used System Programming Language
- Most of the state of the art software have been implemented using C

Why to use C?

C was initially used for system development work, in particular the programs that make-up the operating system. C was adopted as a system development language because it produces code that runs nearly as fast as code written in assembly language. Some examples of the use of C might be:

- Operating Systems
- Language Compilers
- Assemblers
- Text Editors
- Print Spoolers
- Network Drivers
- Modern Programs
- Data Bases
- Language Interpreters
- Utilities

C Program File

All the C programs are written into text files with extension ".c" for example hello.c. You can use "vi" editor to write your C program into a file.

Program Structure: A C program basically has the following form:

- Preprocessor Commands
- Functions
- Variables
- Statements & Expressions
- Comments

The following program is written in the C programming language. Open a text file hello.c using vi editor and put the following lines inside that file.

```
#include <stdio.h>
int main()
{
   printf("Hello, Tech Preparation! \n");
   return 0;
}
```

Preprocessor Commands: These commands tells the compiler to do preprocessing before doing actual compilation. For example #include <stdio.h> is a preprocessor command which tells a C compiler to include stdio.h file before going to actual compilation. You will learn more about C Preprocessors in C Preprocessors session.

Functions: Functions are main building blocks of any C Program. Every C Program will have one or more functions and there is one mandatory function which is called main() function. This function is prefixed with keyword int which means this function returns an integer value when it exits. This integer value is returned using return statement (main function also return void).

The C Programming language provides a set of built-in functions. In the above example printf() is a C built-in function which is used to print anything on the screen.

Variables: Variables are used to hold numbers, strings and complex data for manipulation. You will learn in detail about variables in C Variable Types.

Statements & Expressions: Expressions combine variables and constants to create new values. Statements are expressions, assignments, function calls, or control flow statements which make up C programs.

Comments: Comments are used to give additional useful information inside a C Program. A command can be a multiple line or single line. Multiple line comments will be put inside /*...*/ and single line commands written after //.

Note:

- C is a case sensitive programming language. It means in C printf and Printf will have different meanings.
- C has a free-form line structure. End of each C statement must be marked with a semicolon.
- Multiple statements can be one the same line.
- White Spaces (ie tab space and space bar) are ignored.
- Statements can continue over multiple lines.

C program Compilation

To compile a C program you would have to Compiler name and program files name. Assuming your compiler's name is cc and program file name is hello.c, give following command at Unix prompt.

```
$cc hello.c
```

This will produce a binary file called a.out and an object file hello.o in your current directory. Here a.out is your first program which you will run at Unix prompt like any other system program. If you don't like the name a.out then you can produce a binary file with your own name by using -o option while compiling C program. See an example below

```
$cc -o hello hello.c
```

Now you will get a binary with name hello. Execute this program at Unix prompt but before executing / running this program.

Basic Datatypes

C has a concept of 'data types' which are used to define a variable before its use. The definition of a variable will assign storage for the variable and define the type of data that will be held in the location.

The value of a variable can be changed any time.

C has the following basic built-in datatypes.

- int
- float
- double
- char

Please note that there is not a Boolean data type. C does not have the traditional view about logical comparison, but that's another story.

int - data type: int is used to define integer numbers.

```
{
 int count;
 count = 5;
}
```

float - data type: float is used to define floating point numbers.

```
{
 float Miles;
 Miles = 5.6;
}
```

double - data type: double is used to define BIG floating point numbers. It reserves twice the storage for the number. On PCs this is likely to be 8 bytes.

```
{
 double Atoms;
 Atoms = 2500000;
}
```

char - data type: char defines characters.

```
{
 char Letter;
 Letter = 'x';
}
```

Modifiers

The data types explained above have the following modifiers.

- short
- long
- signed
- unsigned

The modifiers define the amount of storage allocated to the variable. The amount of storage allocated is not cast in stone. ANSI has the following rules:

short int $\Leftarrow$ int $\Leftarrow$ long int

float $\Leftarrow$ double $\Leftarrow$ long double

What this means is that a 'short int' should assign less than or the same amount of storage as an 'int' and the 'int' should be less or the same bytes than a 'long int'. What this means in the real world is:

Type	Bytes	Range	
Short int	2	−32,768 to + 32,767	(32kb)
Unsigned short int	2	0 to + 65,535	(64kb)
Unsigned int	4	0 to + 4,294,967,295	(4Gb)
Int	4	−2,147,483,648 to + 2,147,483,647	(2Gb)
Long int	8	−2,147,483,648 to + 2,147,483,647	(2Gb)
Signed char	1	−128 to + 127	
Unsigned char	1	0 to + 255	
Float	4		
Double	8		
Long double	10		

These figures only apply to today's generation of PCs. Mainframes and midrange machines could use different figures, but would still comply with the rule above.
You can find out how much storage is allocated to a data type by using the sizeof operator discussed in Operator Types Session.
Here is an example to check size of memory taken by various datatypes.

```
int main()
{
printf("sizeof(char) = %d\n", sizeof(char));
printf("sizeof(short) = %d\n", sizeof(short));
printf("sizeof(int) = %d\n", sizeof(int));
printf("sizeof(long) = %d\n", sizeof(long));
printf("sizeof(float) = %d\n", sizeof(float));
printf("sizeof(double) = %d\n", sizeof(double));
printf("sizeof(long double) = %d\n", sizeof(long double));
printf("sizeof(long long) =%d\n", sizeof(long long));

return 0;
}
```

Qualifiers

A type qualifier is used to define the declaration of a variable, a function, and parameters, specifying whether:

- The value of a variable can be changed.
- The value of a variable must always be read from memory rather than from a register

Standard C language recognizes the following two qualifiers:

- const
- volatile

The const qualifier is used to tell C that the variable value cannot change after initialization. const float pi=3.14159;

Now pi cannot be changed at a later time within the program.

Another way to define constants is with the #define preprocessor which has the advantage that it does not use any storage

The volatile qualifier declares a data type that can have its value changed in ways outside the control or detection of the compiler (such as a variable updated by the system clock or by another program). This prevents the compiler from optimizing code referring to the object by storing the object's value in a register and re-reading it from there, rather than from memory, where it may have changed. You will use this qualifier once you will become expert in "C". So for now just proceed.

Variable Types

A variable is just a named area of storage that can hold a single value (numeric or character). The C language demands that you declare the name of each variable that you are going to use and its type, before you actually try to do anything with it.

The Programming language C has two main variable types

- Local Variables
- Global Variables

Local Variables

- Local variables scope is confined within the block or function where it is defined. Local variables must always be defined at the top of a block.
- When a local variable is defined - it is not initialized by the system(garbage value), (Storage class will tell to system during execution what value it will get) initialize it yourself.
- When execution of the block starts the variable is available, and when the block ends the variable 'dies'.

Check following example's output

```
main()
{
int i=4;
int j=10;

i++;

if (j > 0)
{
/* i defined in 'main' can be seen */
printf("i is %d\n",i);
}

if (j > 0)
{
/* 'i' is defined and so local to this block */
int i=100;
printf("i is %d\n",i);
}
/* 'i' (value 100) dies here */
printf("i is %d\n",i); /* 'i' (value 5) is now visable.*/
}
```

This will generate following output

```
i is 5
i is 100
i is 5
```