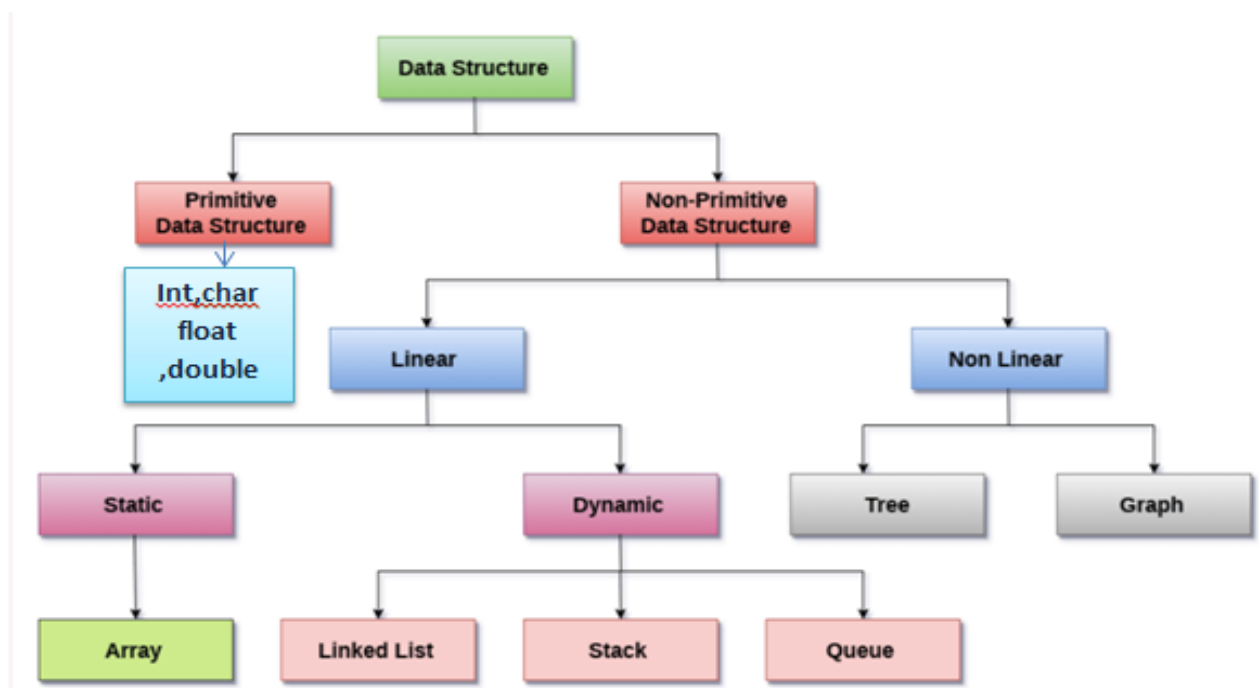


UNIT-1(FUNDAMENTAL NOTATION)

Data Structure:→ Data Structure is a way to store and organize data so that it can be used efficiently. The data structure name indicates itself that organizing the data in memory.

The logical and mathematical model of a particular organization of data is called a datastructure. There are many ways of organizing the data in the memory. Example of Data structures, i.e., array, graph, list etc. In C language..

Types of Data Structures



There are two types of data structures:

- Primitive data structure
- Non-primitive data structure

Primitive Data structure

The primitive data structures are primitive data types. Primitive Data Structures are the basic data structures that directly operate upon the machine instructions.

They have different representations on different computers. The int, char, float, double, and pointer are the primitive data structures that can hold a single value.

Non-Primitive Data structure:→ Non-primitive data structures are more complicated data structures and are derived from primitive data structures.

The non-primitive data structure is divided into two types:

- Linear data structure
- Non-linear data structure

Linear Data Structure

The arrangement of data in a sequential manner is known as a linear data structure. In these data structures, one element is connected to only one another element in a linear form. Linear data structures are easy to implement because computer memory is arranged in a linear way. Its examples are array, stack, queue, linked list, etc.

Non-linear Data Structure:

Data structures where data elements are not arranged sequentially or linearly are called non-linear data structures. In a non-linear data structure, single level is not involved. Therefore, we can't traverse all the elements in single run only. Non-linear data structures are not easy to implement in comparison to linear data structure. It utilizes computer memory efficiently in comparison to a linear data structure. Its examples are trees and graphs.

Linear Data structures can also be classified as:

- **Static data structure:** It is a type of data structure where the size is allocated at the compile time. In Static data structure the size of the structure is fixed. The content of the data structure can be modified but without changing the memory space allocated to it. Therefore, the maximum size is fixed.eg., array.
- **Dynamic data structure:** It is a type of data structure where the size is allocated at the run time. In Dynamic data structure the size of the structure is not fixed and can be modified during the operations performed on it. Dynamic data structures are designed to facilitate change of data structures in the run time. Therefore, the maximum size is flexible.e.g, linked list,queue

Major Operations on DS

The major or the common operations that can be performed on the data structures are:

- **Searching:** We can search for any element in a data structure.
- **Sorting:** We can sort the elements of a data structure either in an ascending or descending order.
- **Insertion:** We can also insert the new element in a data structure.
- **Updation:** We can also update the element, i.e., we can replace the element with another element.
- **Deletion:** We can also perform the delete operation to remove the element from the data structure.
- **Traversing:** Traversing a Data Structure means to visit the element stored in it.
- **Merging:** It is used to combine the data items of two sorted files into single file in the sorted form.

Basic Terminology: Data structures are the building blocks of any program or the software. Choosing the appropriate data structure for a program is the most difficult task for a programmer. Following terminology is used as far as data structures are concerned:

Data: Data can be defined as an elementary value or the collection of values, for example, student's name and its id are the data about the student.

Group Items: Data items which have subordinate data items are called Group item, for example, name of a student can have first name and the last name.

Record: Record can be defined as the collection of various data items, for example, if we talk about the student entity, then its name, address, course and marks can be grouped together to form the record for the student.

File: A File is a collection of various records of one type of entity, for example, if there are 60 employees in the class, then there will be 20 records in the related file where each record contains the data about each employee.

Attribute and Entity: An entity represents the class of certain objects. it contains various attributes. Each attribute represents the particular property of that entity.

Field: Field is a single elementary unit of information representing the attribute of an entity.

Need of data structure

- It gives different level of organization data.
- It tells how data can be stored and accessed in its elementary level.
- Provide operation on group of data, such as adding an item, looking up highest priority item.
- Provide a means to manage huge amount of data efficiently.
- Provide fast searching and sorting of data

Selecting a data structure :→ Selection of suitable data structure involve following steps

- Analyze the problem to determine the resource constraints a solution must meet
- Determine basic operation that must be supported. Quantify resource constraint for each operation
- Select the data structure that best meets these requirements.

Each data structure has cost and benefits. Rarely is one data structure better than other in all situations. A data structure require :

- Space for each item it stores
- Time to perform each basic operation
- Programming effort.

PDLC \Rightarrow Program Development Life Cycle

PDLC is a systematic approach of developing programs. It breaks the job of program development into manageable chunks.

PDLC contains 7 steps

- 1) Problem Definition
- 2) Program Designing.
- 3) Algorithm Development & Flow charting.
- 4) Program Coding.
- 5) Debugging and compilation.
- 6) Program Testing.
- 7) Implementation & Documentation.

1) Problem Definition \Rightarrow This is the very first step to be followed during development of a program. Before writing the program one must have a proper understanding of the problem. He must know what are the o/p requirements of the problem & what type of I/O must be followed (supplied).

2) Program Designing \Rightarrow In this phase its solution procedure is designed and all alternatives are consulted. All types of input format, output

3) Algorithm Development & Flowcharting \Rightarrow Once the design phase is over, the job is totally in the hands of programmer.

Now, programmer prepares the logical design of plan for program using various program design tools, like \rightarrow Algorithm, flow chart, decision table, charts etc.

ii) Program Coding \Rightarrow Normally the language for coding the program for any project is decided in advance.

When the algorithm is developed and tested it needs implementation using that language. This is known as program coding.

5) Debugging & Compilation \Rightarrow Once the program are ready they must be checked for any types of errors.

\rightarrow SYNTAX ERROR

\rightarrow Logical Errors

\rightarrow Execution Errors

Isolation of errors & their removal is known as debugging. compilation is helpful in removing syntax errors.

1) Program Testing → After compilation there may be logical errors which may lead to undesired output.

For testing a program some sample data are taken and their o/p is calculated manually. Then the same data are given as input to the program. The output is compared to the manual o/p. If result is ok, there are no logical errors in the program. After program testing is completed successfully you may move to next step.

2) Implementation & Documentation → This phase consists of major 3 jobs.
→ Installation, → maintenance and documentation

Installation → means to load the program at user site & train him to use the software.

After installation the s/w is evaluated regularly to confirm its ability with time. Also it needs maintenance i.e. there may be some errors discovered by users during its day to day use which need removal and also some times there may be slight change in user requirement which can be made upto date.

Documentation → means collecting, organizing, storing and maintaining complete historical record of the developed s/w. It includes useful content in program, structure chart, Decision table, Algorithm, flowchart etc.

Structured Programming: → Structured programming is concerned with the structures used in a computer program. The programming method used to decompose main function into lower level components for modular coding purpose is called structured programming. In structured programming the program is developed as a series of independent sections designed to perform only one specified task.

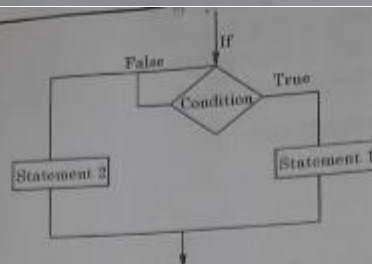
The idea behind the structuring of control flow is to keep program under control. A structure program can be completely developed using four basic control structures.

1. Sequential
2. Conditional
3. Repetition
4. Procedures

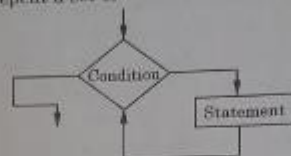
1. **Sequential** : This structure composed of statements executed one after another. There is only one entry point and one exit point.



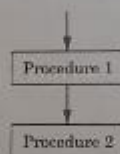
2. **Conditional** : It is also known as selection structure. In this statements are executed depending on certain condition.



3. **Repetitive** : It repeat a set of statements while certain conditions are met.



4. **Procedure** : It enables us to replace a set of statements with a single statement.



The structured programming uses one of the two approaches

- Top down approach
- Bottom up approach

1.3.1 Top Down Approach

It is a disciplined approach used for program designing in which top level functions are decomposed into lower level modules for easy handling and better management. The decomposition is done in a hierarchical manner.

It divides the large problem into smaller problems that can be handled more easily. However if the subproblem is still complex then must be further divided. These subproblem known as module.

Top-down programming focuses on the use of modules. It is therefore also known as modular programming. The program is broken up into small modules so that it is easy to trace a particular segment of code in the software program. The modules at the top level are those that perform general tasks and proceed to other modules to perform a particular task. Top-Down Model is followed by structural programming languages like C, Fortran etc.

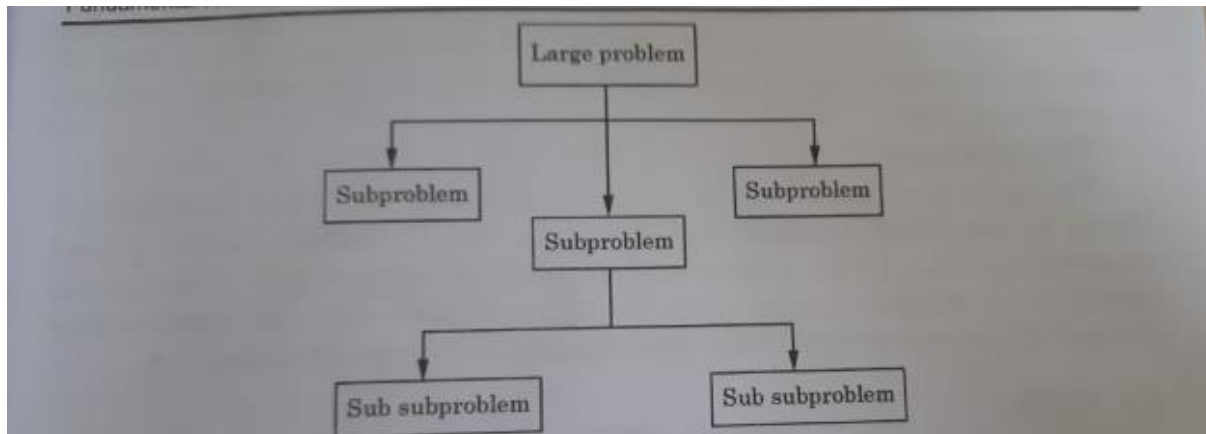


Fig. 1.1 : Top down structure

Disadvantages :

The top down approach is used extensively but it one major drawback. If an error is occurred at some upper level module at later stage then all the modules need changes and refinement at lower levels starting from that level.

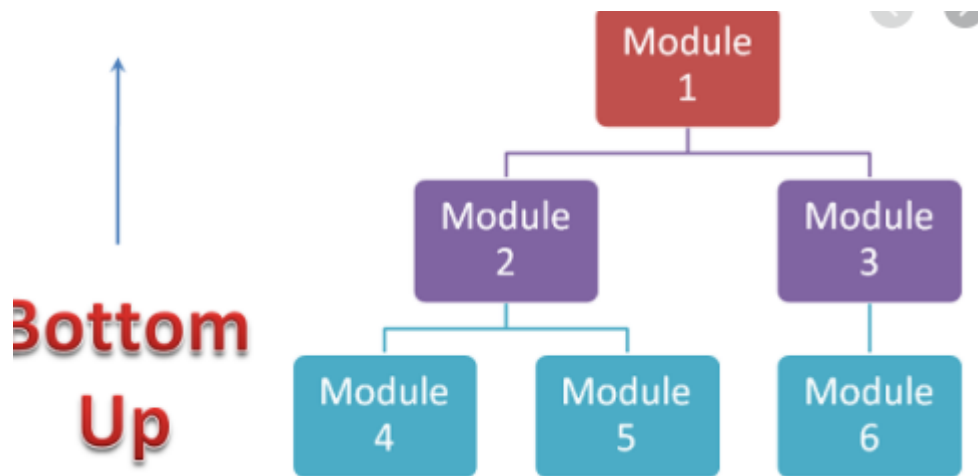
- Top-down model has tight coupling issues and low interactivity between various modules

Advantages→

- By using this approach the program can be developed easily and quickly committing a minimum of error.

Bottom up:→ In this approach instead of starting from top level one start with bottom level modules. The bottom level modules are first prepared and tested and then they combined to go next lowest level modules. Bottom-up programming refers to the style of programming where an application is constructed with the description of modules. The description begins at the bottom of the hierarchy of modules and progresses through higher levels until it reaches the top. Bottom-up programming is just the opposite of top-down programming. Here, the program modules are more general and reusable than top-down programming. Bottom-up model is based on composition approach. Bottom-Up model have high interactivity

between various modules. Bottom-Up Model is mainly used by object oriented programming languages like Java, C++ etc.



Advantages of Structured Programming Approach:

1. Easier to read and understand
2. User Friendly
3. Easier to Maintain
4. Mainly problem based instead of being machine based
5. Development is easier as it requires less effort and time
6. Easier to Debug
7. Machine-Independent, mostly.

Disadvantages of Structured Programming Approach:

1. Since it is Machine-Independent, So it takes time to convert into machine code.
2. The converted machine code is not the same as for assembly language.
3. The program depends upon changeable factors like data-types. Therefore it needs to be updated with the need on the go.
4. Usually the development in this approach takes longer time as it is language-dependent. Whereas in the case of assembly language, the development takes lesser time as it is fixed for the machine.

1.2: Data Type, Variable and Constants:

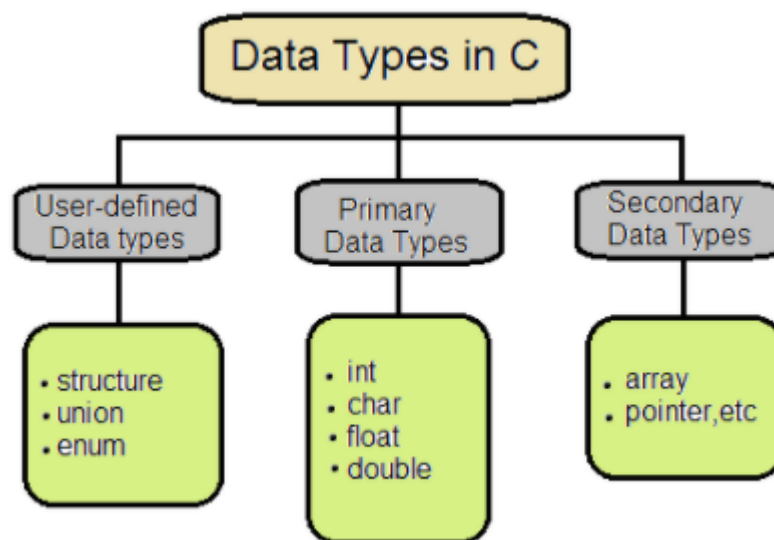
Data and Data Item :→ Data are simply collection of facts and figures. Data are values or set of values.

A data item refers to a single unit of values. Data items that are divided into sub items are group items; those that are not are called elementary items.

For example, a student's name may be divided into three sub items – [first name, middle name and last name] but the ID of a student would normally be treated as a single item.

In the above example (ID, Age, Gender, First, Middle, Last, Street, Area) are elementary data items, whereas (Name, Address) are group data items.

Data Types:→ As its name indicates, a data type represents a type of the data which you can process using your computer program. It can be numeric, alphanumeric, decimal, etc.



Variable:→ A **variable** is a name of the memory location. It is used to store data. Its value can be changed, and it can be reused many times.

It is a way to represent memory location through symbol so that it can be easily identified..

Let's see the syntax to declare a variable:

1. type variable_list;
2. **int** a;
3. **float** b;
4. **char** c;

Here, a, b, c are variables. The int, float, char are the data types.

Rules for defining variables

- A variable can have alphabets, digits, and underscore.

- A variable name can start with the alphabet, and underscore only. It can't start with a digit.
- No whitespace is allowed within the variable name.
- A variable name must not be any reserved word or keyword, e.g. int, float, etc.

Record: → Collection of related data items is known as record. The elements of records are usually called fields or members. Records are distinguished from arrays by the fact that their number of fields is typically fixed, each field has a name, and that each field may have a different type.

Constant: → A constant is a value or variable that can't be changed in the program, for example: 10, 20, 'a', 3.4, "c programming" etc.

There are different types of constants in C programming.

1) . Integer constants: → An integer constant is a numeric constant (associated with number) without any fractional or exponential part. There are three types of integer constants in C programming:

- decimal constant(base 10)
- octal constant(base 8)
- hexadecimal constant(base 16)

Decimal constants: 0, -9, 22 etc

Octal constants: 021, 077, 033 etc

Hexadecimal constants: 0x7f, 0x2a, 0x521 etc

In C programming, octal constant starts with a 0 and hexadecimal constant starts with a 0x.

2) Floating-point constants: → A floating point constant is a numeric constant that has either a fractional form or an exponent form. For example:

2.0
0.0000234
-0.22E-5

E-5 = 10^{-5}

3) Character Constants:-

It is a single character constant enclosed within a single quotation mark (like

'a', 'A'), called a character constants. There are some valid constants as: 'g', 'D', '\n', '#'.

4)String Constant

It is the character set of string constants that are enclosed in a double quote. The character may be letters, numbers, special symbols and some blank space. Furthermore, a string constant contains zero, one or more continuous character in double quotation marks. For example, "Hello Friends", "Computer", "5987", " ", "A".

Note: "A" and 'A' are different; the first one is a string constant consisting of character A and \0. While the second 'A' represents is a character constant whose integer value is 65.

Variables	Constants
It is a variable that stores data type value in a program.	It is similar to a variable and cannot be changed during program execution.
It is a variable that can be changed after defining the variable in a program	It is a fixed variable that cannot be changed after defining the variable in a program.
The value of a variable can change depending on the conditions.	In constants, the value cannot be changed.
Typically, it uses int, float, char, string, double, etc. data types in a program.	It can be express in two ways: #define pre-processor and the const keyword.
Example: int a = 5; float radius = 5.2; char 'A';	Example: const int Len = 5; #define PI 3.14

Pointer: ➡ The pointer in C language is a variable which stores the address of another variable. This variable can be of type int, char, array, function, or any other pointer.

General syntax: ➔ `type *var-name;`

Here, **type** is the pointer's base type; it must be a valid C data type and **var-name** is the name of the pointer variable. The asterisk * used to declare a pointer is the same asterisk used for multiplication. However, in this statement the asterisk is being used to designate a variable as a pointer.

Take a look at some of the valid pointer declarations –

```
int *ip; /* pointer to an integer */
double *dp; /* pointer to a double */
float *fp; /* pointer to a float */
char *ch /* pointer to a character */
```

By the help of * (indirection operator), we can print the value of pointer variable .

Let's see the pointer example as explained for the above figure.

1. `#include<stdio.h>`
2. `int main(){`
3. `int number=50;`
4. `int *p;`
5. `p=&number;` //stores the address of number variable
6. `printf("Address of p variable is %x \n",p);` // p contains the address of the number therefore printing p gives the address of number.
7. `printf("Value of p variable is %d \n",*p);` // As we know that * is used to dereference a pointer therefore if we print *p, we will get the value stored at the address contained by p.
8. `return 0;`
9. `}`

Output

```
Address of number variable is fff4
Address of p variable is fff4
Value of p variable is 50
```

Advantage of pointer

- 1) Pointer **reduces the code** and **improves the performance**, it is used to retrieving strings, trees, etc. and used with arrays, structures, and functions.
- 2) We can **return multiple values from a function** using the pointer.
- 3) It makes you able to **access any memory location** in the computer's memory.

Constant Pointers

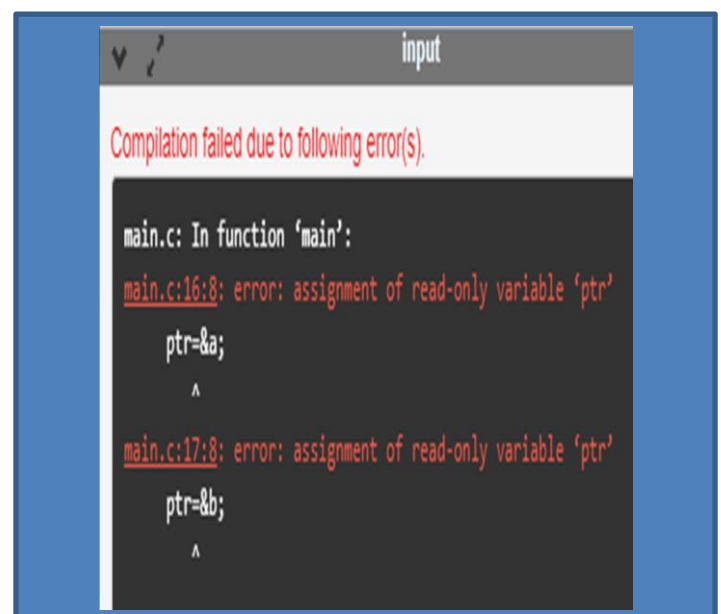
A constant pointer in C cannot change the address of the variable to which it is pointing, i.e., the address will remain constant. Therefore, we can say that if a constant pointer is pointing to some variable, then it cannot point to any other variable.

Syntax of Constant Pointer

1. <type of pointer> *const <name of pointer>;

Declaration of a constant pointer is given below:

1. `int *const ptr;`
 2. `#include <stdio.h>`
 3. `int main()`
 4. `{`
 5. `int a=1;`
 6. `int b=2;`
 7. `int *const ptr;`
 8. `ptr=&a;`
 9. `ptr=&b;`
 10. `printf("Value of ptr is :%d",*ptr);`
 11. `return 0;`
 12. `}`



Pointer to Constant

A pointer to constant is a pointer through which the value of the variable that the pointer points cannot be changed. The address of these pointers can be changed, but the value of the variable that the pointer points cannot be changed.

Syntax of Pointer to Constant

1. **const** <type of pointer>* <name of pointer>

Declaration of a pointer to constant is given below:

```
const int* ptr;
```

1. **#include** <stdio.h>
2. **int** main()
3. {
4. **int** a=100;
5. **int** b=200;
6. **const int*** ptr;
7. ptr=&a;
8. ptr=&b;
9. printf("Value of ptr is :%u",ptr);
10. **return** 0;
11. }

Output

```
Value of ptr is :247760772
```

The above code runs successfully, and it shows the value of 'ptr' in the output.

- Now, we write the code in which we are changing the value of the variable to which the pointer points.

1. **#include** <stdio.h>
2. **int** main()
3. {
4. **int** a=100;
5. **int** b=200;
6. **const int*** ptr;
7. ptr=&b;
8. *ptr=300;
9. printf("Value of ptr is :%d",*ptr);
10. **return** 0;
11. }

