1) // C implementation to demonstrate

// the usage of arrays

#include <stdio.h>

int main()

{

    // Array Indexs-0   1   2   3   4

    int arr[5] = { 10, 20, 30, 40, 50 };

    int size = sizeof(arr);

    printf("Size of array is %d\n", size);

    printf("Data at index 2 is %d\n", arr[2]);

}

**Output**

Size of array is 20

Data at index 2 is 30

Structure is a collection of variables (can be of different data types), under a single name.

It is also a contiguous memory segment like array, but it allows data members of different data types.

**Syntax for variable declaration:**

struct *structure\_name* *variable\_name*;

Example:

struct *structure\_name*

{

datatype *member1\_name;*

datatype *member2\_name;*

*..*

datatype *membern\_name;*

};

|  |
| --- |
| 2) // C implementation to demonstrate  // the usage of structures    #include <stdio.h>  #include <string.h>    // Structure Definition  struct student {      // data members      int roll\_no; // 4 bytes      char name[20]; // 20 bytes  };  int main()  {      // Structure variable Declaration      struct student stu;      stu.roll\_no = 64;      strcpy(stu.name, "Saurabh");      printf("Structure Data\n");      printf("Roll No: %d\n", stu.roll\_no);      printf("Name: %s\n", stu.name);      int size = sizeof(stu);      printf("Size of Structure student");      printf("is %d", size);  } |

**Output**

Structure Data

Roll No: 64

Name: Saurabh

Size of Structure student is 24

* Pointers are the special type of variables that stores the address, rather than the value of the variable.
* They are used for indirect access of variables.
* If var is the name of variable, then ***&var*** gives the address of **var**.

**Syntax for declaration of pointer:**

*data\_type*\* pointer\_name; // (\* = asterisk)

**Example:**

int\* ptr;

3) // C implementation to demonstrate

// pointers in C

#include <stdio.h>

int main()

{

    int\* ptr;

    int num = 5;

    ptr = #

    // This gives address of num

    printf("Value at ptr is %p\n", ptr);

    // This gives value at num

    printf("Value at \*ptr is %d\n", \*ptr);

}

**Output**

Value at ptr is 0x7ffdff4dca9c

Value at \*ptr is 5

**Pointer to the Structure**

* Pointer to the structure can be declared as normal variable.

**Example:**

struct student \*p;

Here,  **p** is pointer and **\*p** is the structure

Hence, to access the data members, we have to use

(\*p).roll\_no

(\*p).name

C provides a special operator for accessing the data members via pointer i.e. -> arrow operator.

**Note:** (\*p).x equivalent to p->x

4. // C implementation to illustrate

// the code of the structures

#include <stdio.h>

#include <stdlib.h>

// Structure Definition

struct student {

    int roll\_no;

    char name[20];

};

int main()

{

    struct student\* p;

    p = (struct student\*)

        malloc(sizeof(struct student));

    // Arrow operator

    p->roll\_no = 99;

    printf("The value at roll");

    printf("number is %d", p->roll\_no);

    return 0;

}

**Output**

The value at rollnumber is 99

## Functions

* A function is a block of code that performs a specific task.
* A function may have an input, performs so tasks, then may provide some output.

**Syntax of function definition:**

*return\_type* function\_name(<parameters\_list>)

{

--tasks/operations--

return *return\_value*;

}

|  |
| --- |
| // C implementation to  // illustrate functions in C    #include <stdio.h>    // program to demonstrate functions  // function definition  // function to print something  void print()  {      printf("GeeksforGeeks\n");  }  // Function to add two numbers  int add(int a, int b)  {      int sum;      sum = a + b;      return sum;  }    // Main Function  int main()  {      int res;      // Function call      print();        res = add(5, 7);        printf("Sum is %d", res);  } |

**Output**

GeeksforGeeks

Sum is 12

**Note:** The type passed in the function call should be compatible with the received by the function body as a parameter. Else, it will cause compilation error.

// C implementation for the

// function call by passing value

#include <stdio.h>

// Function pass by value

void increase\_by\_one(int x)

{

    x++;

}

int main()

{

    int num = 5;

    printf("Value before function");

    printf(" call %d\n", num);

    increase\_by\_one(num);

    printf("Value after function");

    printf(" call %d\n", num);

}

**Output**

Value before function call 5

Value after function call 5

// C implementation to demonstrate

// the usage of function call by

// passing reference

#include <stdio.h>

// function to demonstrate

// call by value

void increase\_by\_one(int\* x)

{

    (\*x)++;

}

int main()

{

    int num = 5;

    printf("Value before function");

    printf(" call %d\n", num);

    increase\_by\_one(&num);

    printf("Value after function");

    printf(" call %d\n", num);

}

**Output**

Value before function call 5

Value after function call 6

// C implementation to demonstrate

// the example of the passing as

// parameter in the function

#include <stdio.h>

// Function to print the array

void print\_array(int arr[], int n)

{

    // N is size of array

    int i;

    for (i = 0; i < n; i++)

        printf("%d ", arr[i]);

}

int main()

{

    int arr[5] = { 10, 20, 30, 40, 50 };

    // Function Call

    print\_array(arr, 5);

}

**Output**

10 20 30 40 50

## Type Casting

Type casting is basically conversion of one datatype into another.

**Syntax of type casting:**

var2 = (datatype2) var1

where,

var1 is of datatype1 & var2 is of datatype2

**Example:**

If you want to convert value of an integer variable into float variable

float x = (float)7/5;

// C program to illustrate the use of

// typecasting

#include <stdio.h>

// Driver Code

int main()

{

    // Given a & b

    int a = 15, b = 2;

    float div;

    // Division of a and b

    div = a / b;

    printf("The result is %f\n", div);

    return 0;

}

**Output:**

The result is 7.000000