```
//This program is used to convert a given number into its / their equivalent
words.
#include<stdio.h>
void main()
{
  int num, i=0, x, d;
  char * word_no [2000];
  printf ("Enter an integer value: \n");
  scanf ("%d", &num);
  while (num)
  {
    d = num %10;
    num = num /10;
    switch(d)
    {
      case 0: word_no[i++] = "zero";
      break;
      case 1: word_no[i++] = "one";
      break;
      case 2: word_no[i++] = "two";
      break;
      case 3: word_no[i++] = "three";
      break;
      case 4: word_no [i++] = "four";
      break;
      case 5: word_no [i++] = "five";
      break;
      case 6: word_no [i++] = "six";
      break;
      case 7: word_no [i++] = "seven";
      break;
      case 8: word_no [i++] = "eight";
      break;
      case 9: word_no[i++] = "nine";
      break;
    }
  }
  for(x=i-1; x>=0; x--){
  printf ("%s ",word_no[x]);
  }
```

}
Practical – II – SY BCA A,B,C -Div- 15 to 19 July 2024

Program – I

C Program to swap a value of 2 variables Using a temporary Variable

This is an example C program uses a temporal variable to swap the values of two variables with each other. This type of program is used quite commonly when developing software.

```
#include <stdio.h>

#include <string.h> /* Header file to support string functions */

void main() {

  /* Declaration of variables used within this program. */

  char first[100], second[100], temp[100];

  /* Take input from the user and save it to the first variable. */

  printf("Enter first string:\n");
  gets(first);

  /* Take input from the user and save it to the second variable. */
  printf("Enter second string:\n");
  gets(second);

  /* Printing the user input before swapping. */
  printf("\nBefore Swapping\n");
  printf("First string: %s\n", first);–
  printf("Second string: %s\n\n", second);

  /* Swapping the variables. */
  strcpy(temp, first); /* Copying the first variable to the flag variable. */
  strcpy(first, second); /* Copying the second variable to the first variable. */
  strcpy(second, temp); /* Copying the flag variable to the second variable. */

  /* Printing the variables after the swapping. */
  printf("After Swapping:\n");
  printf("First string: %s\n", first);
  printf("Second string: %s\n", second);
}
```

Program- II

Illustration of C functions

Check the given number is even or odd

```c
#include <stdio.h>


//if the least significant bit is 1 the number is odd and 0 the
number is even

int checkOddEven(int n1)

{

    return (n1 & 1);//The & operator does a bitwise and,

}


int main()

{

    int n1;

       printf("\n\n Function : check the number is even or odd:\n");

       printf("------------------------------------------------\n");

    printf("Input any number : ");

    scanf("%d", &n1);


    // If checkOddEven() function returns 1 then the number is odd

    if(checkOddEven(n1))

    {

        printf("The entered number is odd.\n\n");

    }

    else

    {

        printf("The entered number is even.\n\n");

    }
    return 0;
}
```

Copy
Sample Output:

```
 Function : check the number is even or odd:
------------------------------------------------
Input any number : 5
The entered number is odd.
```

Explanation:

```
int checkOddEven(int n1) {
   return (n1 & 1); //The & operator does a bitwise and,
 }
```

The above function 'checkOddEven' takes a single argument of type int, named 'n1'. It checks whether the value of 'n1' is odd or even by performing a bitwise AND operation between 'n1' and the binary value 1.

If the result of this operation is 1, then the least significant bit of n1 is 1, which means that n1 is odd.

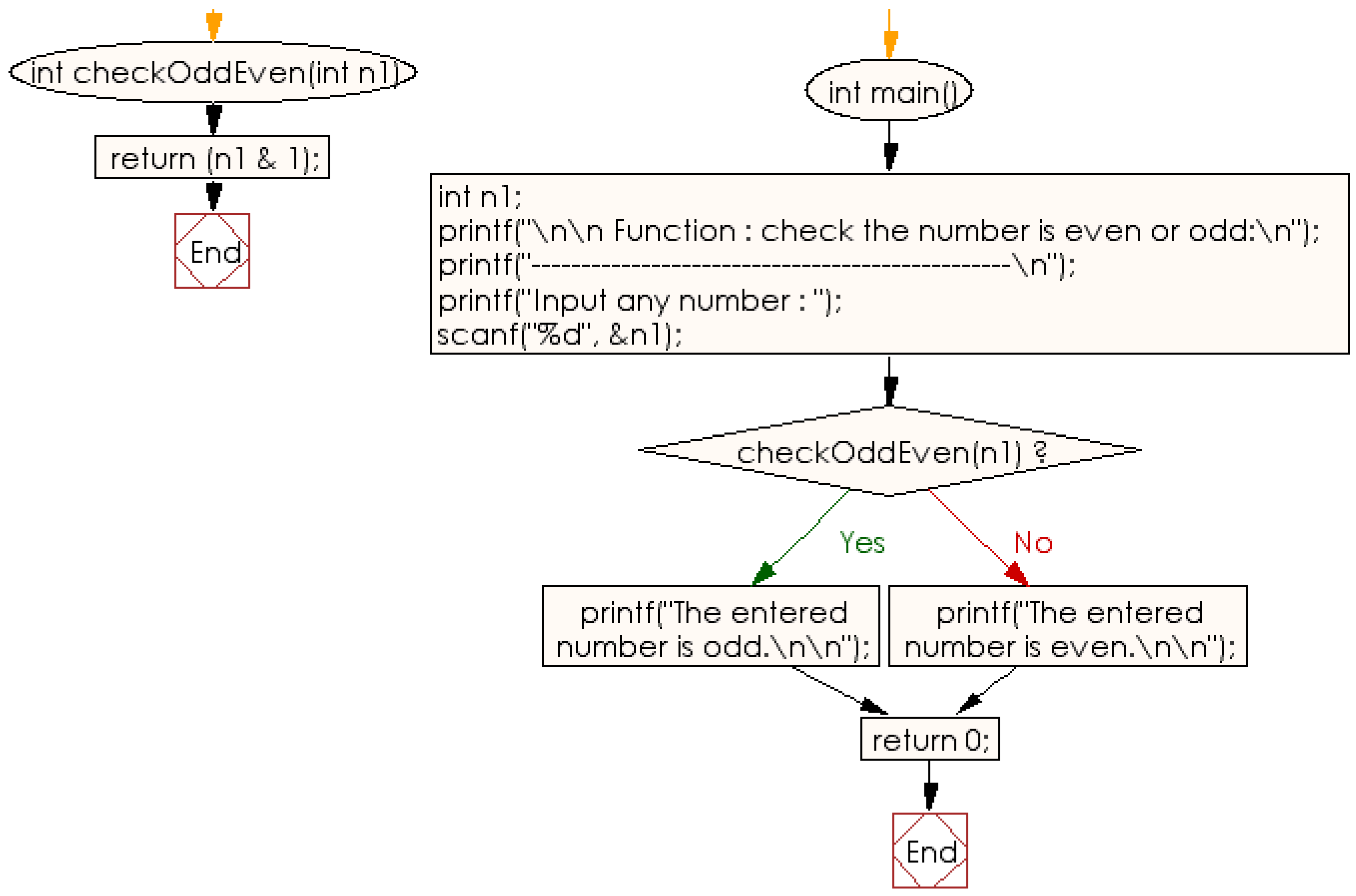
If the result is 0, then the least significant bit of n1 is 0, which means that n1 is even. The function then returns the result of as an integer value of 0 (even) or 1 (odd).

Time complexity and space complexity:

The time complexity of the function ‘checkOddEven’ is Big O(1), as the computation it performs is constant and independent of the input size.

The space complexity of this function is also Big O(1), as it only uses a fixed amount of memory to store the single integer variable n1.

Flowchart:


int checkOddEven(int n1)
return (n1 & 1);
End
int main()
int n1;
printf("\n\n Function : check the number is even or odd:\n");
printf("--\n");
printf("Input any number : ");
scanf("%d", &n1);
checkOddEven(n1) ?
Yes
No
printf("The entered number is odd.\n\n");
printf("The entered number is even.\n\n");
return 0;
End

Program - III

Get largest element of an array

;.

```
#include<stdio.h>

#define MAX 100

int findMaxElem(int []);

int n;

int main()

{    int arr1[MAX],mxelem,i;

       printf("\n\n Function : get largest element of an array
:\n");


printf("-------------------------------------------------\n");
```

```
       printf(" Input the number of elements to be stored in the
array :");
       scanf("%d",&n);


       printf(" Input %d elements in the array :\n",n);
       for(i=0;i<n;i++)
        {
             printf(" element - %d : ",i);
             scanf("%d",&arr1[i]);
          }
    mxelem=findMaxElem(arr1);


    printf(" The largest element in the array is : %d\n\n",mxelem);
    return 0;
}
int findMaxElem(int arr1[])
{
    int i=1,mxelem;
    mxelem=arr1[0];
    while(i < n)
      {
      if(mxelem<arr1[i])
           mxelem=arr1[i];
      i++;
    }
    return mxelem;
}
```

Sample Output:

```
Function : get largest element of an array :
-------------------------------------------------
 Input the number of elements to be stored in the array :5
 Input 5 elements in the array :
 element - 0 : 1
 element - 1 : 2
 element - 2 : 3
 element - 3 : 4
 element - 4 : 5
 The largest element in the array is : 5
```

Explanation:

```
int findMaxElem(int arr1[]) {
  int i = 1, mxelem;
  mxelem = arr1[0];
  while (i < n) {
    if (mxelem < arr1[i])
      mxelem = arr1[i];
    i++;
  }
  return mxelem;
}
```

The function 'findMaxElem' and takes a single argument of type int array, named arr1. It finds the maximum element in the array by iterating through all elements of the array and keeping track of the maximum element seen so far.

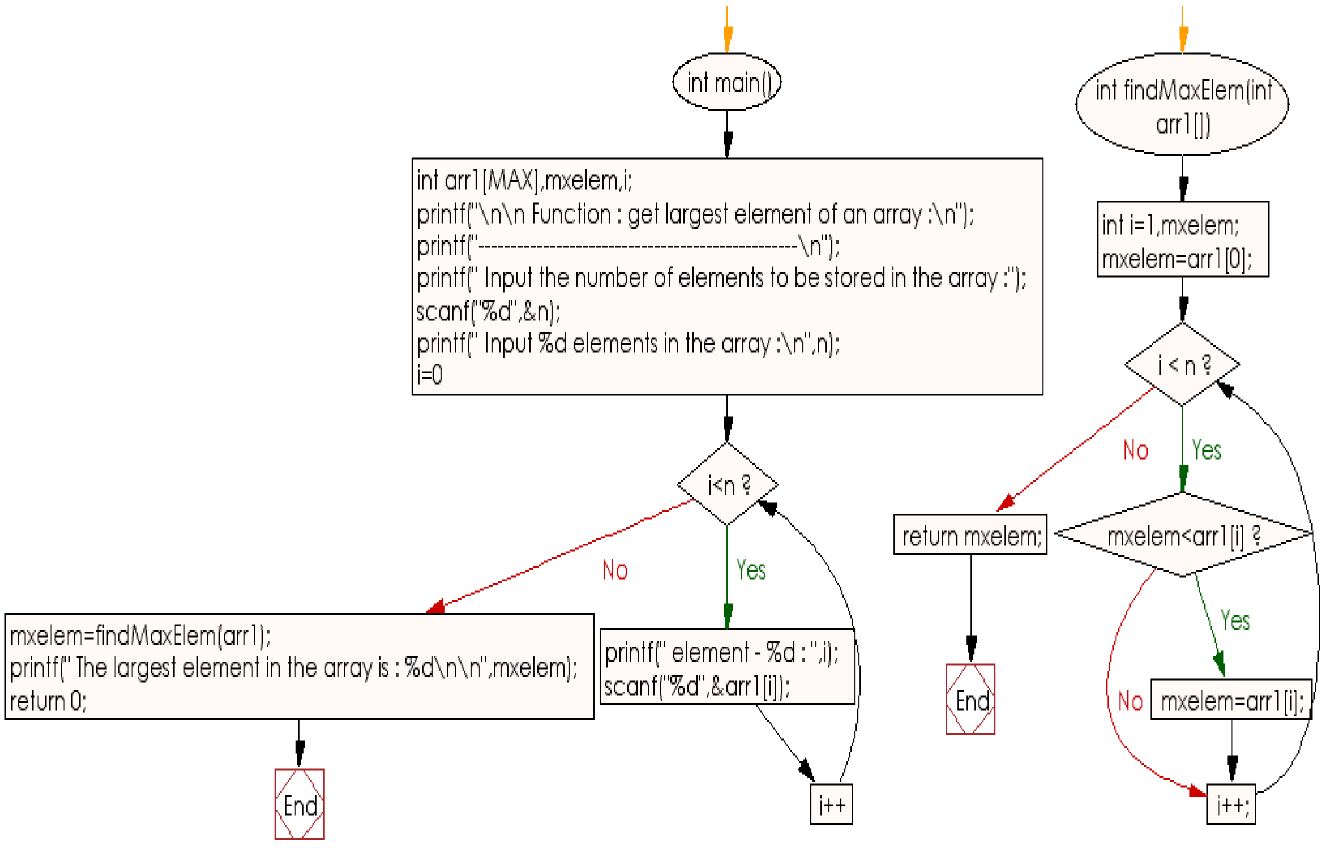
The function initializes two local integer variables, i and mxelem, to 1 and the first element of the array arr1[0] respectively. It then enters a while loop that continues as long as i is less than the length of the array n.

In each iteration of the loop, the function compares the current element of the array arr1[i] with the current maximum element mxelem. If arr1[i] is greater than mxelem, mxelem is updated to the value of arr1[i]. The loop continues until all elements of the array have been processed. Finally, the function returns the maximum element 'mxelem'.

Time complexity and space complexity:

The time complexity of this function is O(n), where n is the length of the input array arr1, because the while loop iterates n-1 times to find the maximum element of the array. The space complexity of this function is O(1), as it only uses a fixed amount of memory to store the three integer variables i, mxelem, and the input array arr1.

Flowchart:


int main()
int arr1[MAX],mxelem,i;
printf("\n\n Function : get largest element of an array :\n");
printf("--\n");
printf(" Input the number of elements to be stored in the array :");
scanf("%d",&n);
printf(" Input %d elements in the array :\n",n);
i=0
i<n ?
No
Yes
mxelem=findMaxElem(arr1);
printf(" The largest element in the array is : %d\n\n",mxelem);
return 0;
End
printf(" element - %d : ",i);
scanf("%d",&arr1[i]);
i++
int findMaxElem(int arr1[])
int i=1,mxelem;
mxelem=arr1[0];
i < n ?
No
Yes
return mxelem;
End
mxelem<arr1[i] ?
Yes
No
mxelem=arr1[i];
i++;

Program – IV

Sum of Natural Numbers Using Recursion

```c
#include <stdio.h>

int addNumbers(int n);

int main() {

  int num;
  printf("Enter a positive integer: ");
  scanf("%d", &num);
  printf("Sum = %d", addNumbers(num));
  return 0;
}

int addNumbers(int n) {
  if (n != 0)
    return n + addNumbers(n - 1);
  else
```

```
    return n;
}
```
Run Code

Output

```
Enter a positive integer: 20
Sum = 210
```

Suppose the user entered **20**.
Initially, `addNumbers()` is called from `main()` with **20** passed as an argument.
The number **20** is added to the result of `addNumbers(19)`.
In the next function call from `addNumbers()` to `addNumbers()`, **19** is passed which is added to the result of `addNumbers(18)`. This process continues until `n` is equal to **0**.
When `n` is equal to **0**, there is no recursive call. This returns the sum of integers ultimately to the `main()` function.

Program – V

C Program to Store Information of a Student Using Structure

To understand this example, you should have the knowledge of the following C programming topics:

- C struct

Store Information and Display it Using Structure

```
#include <stdio.h>
```

```c
struct student {
    char name[50];
    int roll;
    float marks;
} s;

int main() {
    printf("Enter information:\n");
    printf("Enter name: ");
    fgets(s.name, sizeof(s.name), stdin);

    printf("Enter roll number: ");
    scanf("%d", &s.roll);
    printf("Enter marks: ");
    scanf("%f", &s.marks);

    printf("Displaying Information:\n");
    printf("Name: ");
    printf("%s", s.name);
    printf("Roll number: %d\n", s.roll);
    printf("Marks: %.1f\n", s.marks);

    return 0;
}
```
Run Code

Output

```
Enter information:
Enter name: Jack
Enter roll number: 23
Enter marks: 34.5
Displaying Information:
Name: Jack
Roll number: 23
Marks: 34.5
```

In this program, a structure `student` is created. The structure has three members: `name` (string), `roll` (integer) and `marks` (float).
Then, a structure variable `s` is created to store information and display it on the screen.

Share on:

Did you find this article helpful?

Related Examples

C Example