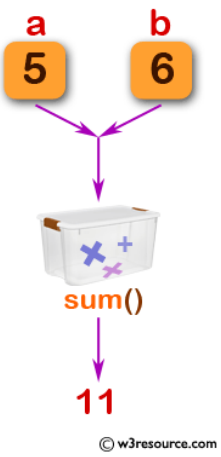


User Created Functions in C

In most programming languages the user can write their own functions to perform specific tasks. Once created, they can be **called** into a program just like the C's built-in functions. **User composed Functions** are useful if you need to complete a complex task repeatedly. Commonly, larger programs are divided up into several individual functions. This is a great way to organise your work and assist with debugging.

Defining a Function in C

The general form of a function definition in C:

```
return_type  function_name ( parameter list )
 {
   body of the function
 }
```

Return Type – A function may return a value. The **return_type** is the data type of the value the function returns (`int char, float etc`). Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword **void**.

Function Name – This is the actual name of the function.

Parameters – A parameter is like a placeholder. When a function is invoked, **you pass a values into it**. The value of a parameter can be called an argument. Some functions contain no parameters.

Function Body – The function body contains a collection of statements that define what the function does.

Example:

```
int sum (int b, int a) //function definition
{
     int s;
     s=a+b;
    return s;       //function returning a value
}
```

Calling the function in a program:

```
int sum (int b, int a) //function definition
{
     int s;
     s=a+b;
      return s;       //function returning a value
}

int main (void)
{
    int total;
    int num1, num2;
    printf("enter a number:\n");
    scanf("%d", &num1);
    printf("enter a second number and I will add them:\n");
    scanf("%d", &num2);
    total = sum (num1, num2); //function is called
    printf ("The total is :  %d\n", total);
    return 0;
}
```

New Syntax

Notice: `int main()` has been replaced with `int main(void)`. In most cases either one is fine, but technically `int main()` is used when you don't need to put arguments into the main program. `int main(void)` can **only** be called *without* any arguments.

Exercise 5.1

Copy and paste the code above and run in your online compiler. You can try getting rid of the void if you like…nothing should change.

Exercise 5.2 (You try and check)

Write a **function** in C to find the square of any number run your function within an `int main()` program.

Example Input/ouput:

Input any number to be squared: 20

The square of 20 is: 400.00

Sample solutions below:

```c
#include <stdio.h>

double square(double num)
{
    return (num * num);
}
int main()
{
    int num;
    double n;
    printf("Input any number to be squared: ");
    scanf("%d", &num);
    n = square(num);
    printf("The square of %d is : %.2f\n", num, n);
    return 0;
}
```

Exercise 5.3 (Type it in...or try if you wish)

Write a function in C to get largest element of an array. Run your function within an `int main()` program.

Sample input:

2
56
7
8
87
32

Sample output:

87

Sample Solution: (note that code continues on a second page)

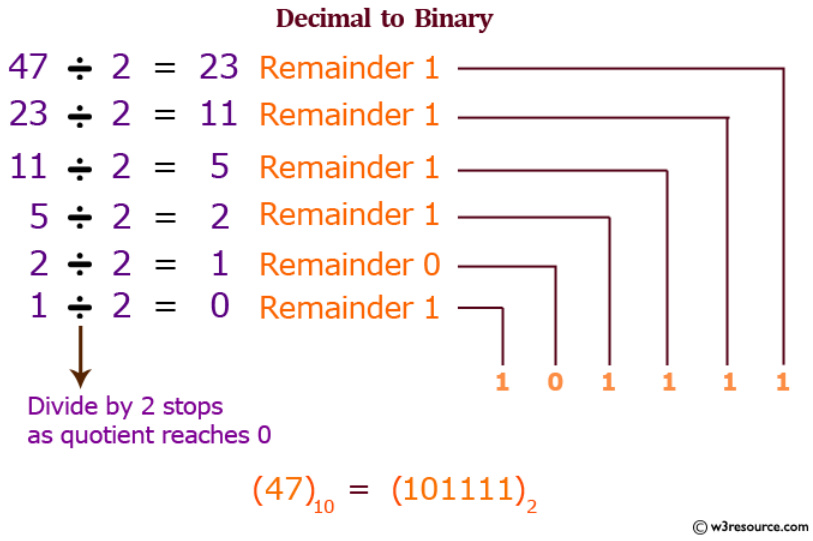
```
#include<stdio.h>
#define MAX 100

int findMaxElem(int []);
int n;

int main()      MAIN PROGRAM DECLARED
{
    int arr1[MAX],biggest,i;

       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]);
         }
    biggest=findMaxElem(arr1); function findMaxElem Called in main program

    printf(" The largest element in the array is : %d\n\n",biggest);
    return 0;
}
```

Code continued on next page…..

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

Exercise 5.4 (try if you like – **intended to be typed in**)

Write a program in C to convert decimal number to binary number using the function.



Sample Input/Output:

```
Input any decimal number: 65

The Binary value is: 1000001
```

Solution on NEXT PAGE

Sample solution:

```c
#include<stdio.h>

long toBin(int);

int main()
{
    long bno;
    int dno;
      printf("\n\n Function : convert decimal to binary :\n");
      printf("-------------------------------------------\n");
    printf(" Input any decimal number : ");
    scanf("%d",&dno);
    bno = toBin(dno);
    printf("\n The Binary value is : %ld\n\n",bno);

    return 0;
}
long toBin(int dno)
{
    long bno=0,remainder,f=1;
    while(dno != 0)
    {
         remainder = dno % 2;
         bno = bno + remainder * f;
         f = f * 10;
         dno = dno / 2;
    }
    return bno;
}
```

Exercise 5.5 (Your try on your own)

Create a program that can perform basic calculations (add, subtract, multiply, divide). Use a menu at the beginning of your program. You must create separate functions for each operation. Each function should be called in the program when required. **You may use old code from a previous assignment to get you started.**

Global vs. Local Variables

Variables can be declared either i**nside** or **outside** a function:

Global Variables (declared outside – affect everything!)

Global variables are declared **outside** any function, and they can be accessed (used) in any function in the program.

Local Variables (declared inside – affect only that function)

Local variables are declared **inside** a function and are available only to that specific function. They are created when the function is called, and destroyed after the function returns. They are called ***local variables*** because they are local to the function and not available outside the function.

```
#include <stdio.h>

int a = 20;  ← Global variable

int main ()

{
  int a = 10;  ← Local variable
  int b = 20;
  int c = 0;

  printf ("value of a in main() = %d\n",  a);
  c = sum( a, b);
  printf ("value of c in main() = %d\n",  c);
  return 0;
}

int sum(int a, int b)
 {

   printf ("value of a in sum() = %d\n",  a);
   printf ("value of b in sum() = %d\n",  b);

   return a + b;
```

Variable Scope (important!)

A **global variable** is **seen by the whole program**, **local variables** are **only seen by the function in which they are contained**;

However, if a local variable has the **same name** as a global variable, the local variable will take precedent (the local variable "wins" and gets used by the function! Not the global variable)

To avoid confusion, best **not to have two variables with the same name!**

Example:

```
#include <stdio.h>

int a = 20;      Global

int main ()
{
  int a = 10;    Local
  int b = 20;
  int c = 0;

  printf ("value of a in main() = %d\n",  a);
  c = sum( a, b);
  printf ("value of c in main() = %d\n",  c);
  return 0;
}
int sum(int a, int b)
 {

   printf ("value of a in sum() = %d\n",  a);
   printf ("value of b in sum() = %d\n",  b);

   return a + b;
 }
```

In the case above:

```
value of a in main() = 10
value of a in sum() = 10
value of b in sum() = 20
value of c in main() = 30
```

Read the above code carefully, then cut and paste it into the compiler.