Functions

Arash Rafiey

September 26, 2017

Functions

Functions are the basic building blocks of a C program.

Functions

Functions are the basic building blocks of a C program.

A **function** can be defined as a set of instructions to perform a specific task.

Functions

Functions are the basic building blocks of a C program.

A **function** can be defined as a set of instructions to perform a specific task.

In C functions provide **reusability** of code and improve **understandability** of large programs.

Function Definition

Syntax for function definition:

return_type function_name (parameter list)

{

functon body

}

Function Definition

Syntax for function definition:

return_type function_name (parameter list)

{

functon body

}

- **Return type:**
  Functions can return values. return_type is the data type of the value which the function returns.

Function Definition

Syntax for function definition:

return_type function_name (parameter list)

{

functon body

}

- **Return type:**
  Functions can return values. return_type is the data type of the value which the function returns.

  If the function does not return any value, then return_type is **void**

Function Definition

- **Parameter list:**
  Parameter list are the parameters (or arguments) that are passed to the function when it is called.

Function Definition

- **Parameter list:**
  Parameter list are the parameters (or arguments) that are passed to the function when it is called.

  It consists of parameter_type and parameter_name.

  A function may contain no parameters.

Function Definition

- **Parameter list:**
  Parameter list are the parameters (or arguments) that are passed to the function when it is called.

  It consists of parameter_type and parameter_name.

  A function may contain no parameters.

- **Function body:** The function body consists of statements that perform a specific task.

Example

Example: Simple function to find the maximum of two numbers.

Example

Example: Simple function to find the maximum of two numbers.

```
int max ( int a, int b )
{
 if ( a<b )
   return b;
 else
   return a;
}
```

Function Declaration

Function declaration tells the compiler about the function name, return_type and the parameters.

Function Declaration

Function declaration tells the compiler about the function name, return_type and the parameters.

Syntax:

return_type function_name (parameter list);

Function Declaration

Function declaration tells the compiler about the function name, return_type and the parameters.

Syntax:

return_type function_name (parameter list);

Example:

```
int sum ( int a, int b);
```

Function Call

A function performs its specified task when it is called by the program.

Function Call

A function performs its specified task when it is called by the program.

Syntax to call a function:

function_name (parameter(1),...parameter(n));

Function Call

A function performs its specified task when it is called by the program.

Syntax to call a function:

function_name (parameter(1),...parameter(n));

Example:

sum(a, b);

Function Call

A function performs its specified task when it is called by the program.

Syntax to call a function:

function_name (parameter(1),...parameter(n));

Example:

sum(a, b);

If the function returns some value then we can store the return value;

Function Call

A function performs its specified task when it is called by the program.

Syntax to call a function:

function_name (parameter(1),...parameter(n));

Example:

sum(a, b);

If the function returns some value then we can store the return value;

Example:

```
int x = sum ( a, b );
```

Example

Example: Program to find factorial of a number.

```
# include <stdio.h>
long int fact(int);
void main() {
int i,n;
long int factorial;
   printf(" Enter a number \n");
   scanf("%d", &n);
   factorial = fact( n );
   printf(" Factorial is: %ld", factorial );
}
```

Example

```
long int fact( int num ) {
  int i;
  long int f=1;
   for (i = 1; i<num; i++)
    f = f *i;
  return f;
}
```

Sorting an Array

```
# include <stdio.h>
void sort(int a, int numbers[ ] );
void main() {
 int i, n;
 printf("Enter number of elements in array \n");
 scanf("%d",&n);
 int arr[n];
 printf("Enter the elements of the array \n");
  for (i = 0; i<n; i++)
   scanf("%d", &arr[i]);
 sort( n, arr );
}
```

Sorting an Array

```
void sort( int a, int numbers[ ] ) {
int i, j, temp;
for (i = 0; i<a; i++){
   for (j = i+1; j<a; j++) {
    if (numbers[i]> numbers[j]){
     temp = numbers[i];
     numbers[i] = numbers[j];
     numbers[j] = temp; }
  }
}
printf("The numbers in ascending order are: \n");
for (i = 0; i<a; ++i)
printf("%d \n", numbers[i]);
}
```

Function call

When a function is called, the parameters can be passed in two ways.

- **Call by value**
- **Call by reference**

Call by value

- **Call by value:**

  In this method the actual value of the parameter is copied into the formal parameter of the function.

Call by value

- **Call by value:**

  In this method the actual value of the parameter is copied into the formal parameter of the function.

  Formal parameter is the value which is used in the function definition.

Call by value

- **Call by value:**

  In this method the actual value of the parameter is copied into the formal parameter of the function.

  Formal parameter is the value which is used in the function definition.

  Any changes to the formal parameters inside the function does not affect the actual value of the parameters.

Call by value

- **Call by value:**

  In this method the actual value of the parameter is copied into the formal parameter of the function.

  Formal parameter is the value which is used in the function definition.

  Any changes to the formal parameters inside the function does not affect the actual value of the parameters.

  In C, call by value is used by default.

Call by value

Example: Program to swap two numbers using **call by value**

```
# include<stdio.h>
void swap( int a, int b);
void main() {
  int x = 10, y = 20;
  printf("Values before swap in main x = %d, y = %d \n",x,y);
  //x=10 y=20
  swap(x,y);
  printf("Values after swap in main x = %d, y = %d \n",x,y);
  //x=10 y=20
}
```

Call by value

```
void swap( int a, int b )
{
  int temp;
  temp = a;
  a = b;
  b = temp;
  printf(" Values after swap inside swap function: x = %d, y = %d", a, b);
  //x=20 y=10
}
```

Call by reference

- **Call by reference:**

  In this method the address of the actual parameter is copied into the formal parameter of the function.

Call by reference

- **Call by reference:**

  In this method the address of the actual parameter is copied into the formal parameter of the function.

  Actual parameter is the parameter which is used in the function call.

Call by reference

- **Call by reference:**

  In this method the address of the actual parameter is copied into the formal parameter of the function.

  Actual parameter is the parameter which is used in the function call.

  Changes to the formal parameters changes the value of the actual parameters.

Call by reference

Example: Program to swap two numbers using **call by reference**

```
# include<stdio.h>
void swap( int *a, int *b);
void main() {
  int x = 10, y = 20;
  printf("Values before swap in main x = %d, y = %d \n",x,y);
  //x=10 y=20
  swap(&x,&y);
  printf("Values after swap in main x = %d, y = %d \n",x,y);
  //x=20 y=10
}
```

Call by reference

```
void swap( int *a, int *b )
{
 int temp;
 temp = *a; // the value store in "a" is assigned to temp
 *a = *b;
// here "a" is an address. Exchange the values of the addresses of
// "a", "b"
 *b = temp;
 printf(" Values after swap inside swap function: x = %d, y =
%d", *a, *b);
 //x=20 y=10
}
```

Tutorial for Thursday September 28

Question (1 by Lavanya)

Write a C program to calculate the square of a given number using function concepts. function name: "square"

Under main() function:

1. Read an integer "n"
2. pass "n" to the function square
3. print the output

Under square(int n) function:

1. calculate the square of n
2. Store it under a variable "res"
3. return "res"

Sample Output:

Enter the number:

5

Square of given number 5:

$5^2 = 25$

Question (2)

Write a program that reads n integer numbers from the user and then pass them to a function reverse which prints them in the reverse order.

Hint : You need to declare and array and store the numbers in that array then pass the array to the reverse fuction which it goes from the beginning to the end of the array and print the elements of the array. (See the slides 23,24).

Question (3 by Lavanya)

Write a C program using switch-case: if input = A, perform addition operation on given two numbers and return the output
if input = S, perform subtraction operation on given two numbers and return the output
if input = D, perform division operation on given two numbers and return the output
if input = E, print "No operation selected" and exit

Sample Output: Press "A" for addition, "S" for subtraction, "D" for division, "E" to exit
A
5
10
Addition operation: $5 + 10 = 15$
See slide 22
http://cs.indstate.edu/~arash/teaching/CS256lec3.pdf

Question (4)

Write a program that reads two strings A1, A2 from the user and send them to a function "compare-strings" that checks whether they are the same or not.

Hint :

```
bool compare-strings (char A1[], char A2[]);
int main() {
bool flag; // boolean variable
char A1[30];
char A2[30];
scanf("%s", &A1);
scanf("%s", &A2);
flag=compare-strings(A1,A2);
if ( flag ) printf("the same");
else printf ("different");
}
```

```
bool compare-string (char A1[], char A2[] ) {
int len1=0;
int len2=0;
}
```

Question (5)

What does this code print ?

```
int main() {
int x=20;
int *p;
p = &x;
*p = 30;
printf("x is %d ",x);
}
```

Question (6)

Write a program that reads two sorted lists (sorted arrays) $L1, L2$ of lengths n, m (respectively) and merge them into a new sorted list $L3$.

Sample Input:
$L1 : 2, 3, 7, 10, 21, 37$ and $L2 : 1, 2, 9, 45$

Output :
$L3 : 1, 2, 2, 3, 7, 9, 10, 21, 37, 45$