Fundamentals of Computing: Lecture 10

Piyush P Kurur
Office no: 224
Dept. of Comp. Sci. and Engg.
IIT Kanpur

August 19, 2009

Summary of Function

- A function takes few arguments and returns a value.

Summary of Function

- A function takes few arguments and returns a value.
- The function declaration tells us/compiler what are the types of the argument of the function and what is its return type.

Summary of Function

- A function takes few arguments and returns a value.
- The function declaration tells us/compiler what are the types of the argument of the function and what is its return type.
- The function definition says what the function actually does.

Summary of Function

- A function takes few arguments and returns a value.
- The function declaration tells us/compiler what are the types of the argument of the function and what is its return type.
- The function definition says what the function actually does.
- C follows call be value and hence change in the argument of the function does not effect the callee.

Summary of Function

- A function takes few arguments and returns a value.
- The function declaration tells us/compiler what are the types of the argument of the function and what is its return type.
- The function definition says what the function actually does.
- C follows call be value and hence change in the argument of the function does not effect the callee.
- Function can call other functions even itself.

Example of a function declaration

```
   void hanoi(int, char, char,char);
```

or

```
   void hanoi(int, char a, char b, char c);
```

Example of a function definition.

```
 void hanoi(int n, char src, char inter, char dest)
 {
         if ( n <= 0 ) return;
         hanoi(n-1 , src, dest, inter);
         printf("(%d) %c -> %c\n", n, src, dest);
         hanoi( n-1, inter, src, dest);
 }


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

Function calls are returns

When a function is called

```
/* do some thing */
foo(2+4, y);
/* do something else */
```

- Control goes to the begining of the functions.

Function calls are returns

When a function is called

```
/* do some thing */
foo(2+4, y);
/* do something else */
```

- Control goes to the begining of the functions.
- Fresh variables are created for each parameters whose value is the respective value when called.

Function calls are returns

When a function is called

```
/* do some thing */
foo(2+4, y);
/* do something else */
```

- Control goes to the begining of the functions.
- Fresh variables are created for each parameters whose value is the respective value when called.

When `return` is encountered

- The control goes back to where it was called.

When `return` is encountered

- The control goes back to where it was called.
- If the return is of the form `return expr` the program behaves as if the value of `expr` is substituted in the place where the function is called.

When `return` is encountered

- The control goes back to where it was called.
- If the return is of the form `return expr` the program behaves as if the value of `expr` is substituted in the place where the function is called.

```
printf("%d", fact(3));
/* some more stuff */

int fact(n)
{
  if (n < 2) return 1;
  else return n * fact(n-1);
}
```

Variable declaration and scope

```
#include <stdio.h>

int global=0;
void foo(int t);
int main()
{
  printf("in main global = %d\n", global);
  foo(0); global = 42; foo(1);
  int global = 100;
  printf("in main after dec global = %d\n",global);
  foo(2); global=10; foo(3);
  printf("in main after dec and update global = %d\n",glob
}
void foo(int t)
{
  int local = 120;
  printf("in foo(%d) global = %d, local = %d\n", t, globa
}
```

Variable scope

- A variable comes to life when it is declared.
- A variable lives as long as the smallest block that contains its declartion is active
- A variable outside every functions is global and lives forever.
- Local variables have precedence over global ones.

Variables in for loop

```
for(int i = 0; i < 100; i++)
{
  /* do something */
}
```

Variables in for loop

```
for(int i = 0; i < 100; i++)
{
  /* do something */
}
```

The variable `i` is valid only within the for loop.

Variables inside function

```
int foo(int x)
{
  /* some stuff */
  float local;

  foo(bar);

}
```

Variables inside function

```
int foo(int x)
{
  /* some stuff */
  float local;

  foo(bar);

}
```

- The variable is local to the function.

Variables inside function

```
int foo(int x)
{
  /* some stuff */
  float local;

  foo(bar);

}
```

- The variable is local to the function.
- For a new call of foo there is a new variable named local valid for that called