Fundamentals of Computing: Lecture 7

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

August 10, 2009

Summary of last class

Summary of last class

- We studied arrays,

Summary of last class

- We studied arrays,
- Character type and String type (character array).

Summary of last class

- We studied arrays,
- Character type and String type (character array).
- `printf` and `scanf`.

Summary of last class

- We studied arrays,
- Character type and String type (character array).
- `printf` and `scanf`.

Functions

Motivation

Functions

Motivation

- Program fragments are used may times

Functions

Motivation

- Program fragments are used may times
- Better organisation of code (refactoring).

Functions

Motivation

- Program fragments are used may times
- Better organisation of code (refactoring).

Function definition

```
type function-name(arg1,arg2,...,argn) statement-block
```

Functions

Motivation

- Program fragments are used may times
- Better organisation of code (refactoring).

Function definition

`type function-name(arg1,arg2,...,argn) statement-block`

eg.

```
double square(double x)
{
  return x * x;
}
```

A complete example

```
# include <stdio.h>
 void swap(int, int); /* declaration */
 int main()
 {
   int x = 15, y = 42;
   printf("x = %d, y = %d\n", x, y);
   swap(x,y);
   printf("x = %d, y = %d\n", x, y);
 }

 void swap(int u, int v)/* definition */
 {
   int temp;
   temp = u;
   u = v;
   v = temp;
   return;
 }
```

Argument passing scheme: Call by value

C follows the *call by value* argument passing scheme.

Argument passing scheme: Call by value

C follows the *call by value* argument passing scheme.

- Only values are passed to the arguments,
- Changes to parameters in the function does *not* affect the callee

Argument passing scheme: Call by value

C follows the *call by value* argument passing scheme.

- Only values are passed to the arguments,
- Changes to parameters in the function does *not* affect the callee

Call by reference

Argument passing scheme: Call by value

C follows the *call by value* argument passing scheme.

- Only values are passed to the arguments,
- Changes to parameters in the function does *not* affect the callee

Call by reference

- C has only call by value.

Argument passing scheme: Call by value

C follows the *call by value* argument passing scheme.

- Only values are passed to the arguments,
- Changes to parameters in the function does *not* affect the callee

Call by reference

- C has only call by value.
- Fortran has only call be reference.

Argument passing scheme: Call by value

C follows the *call by value* argument passing scheme.

- Only values are passed to the arguments,
- Changes to parameters in the function does *not* affect the callee

Call by reference

- C has only call by value.
- Fortran has only call be reference.
- Pascal and C++ has both call be value and reference.

Argument passing scheme: Call by value

C follows the *call by value* argument passing scheme.

- Only values are passed to the arguments,
- Changes to parameters in the function does *not* affect the callee

Call by reference

- C has only call by value.
- Fortran has only call be reference.
- Pascal and C++ has both call be value and reference.
- Java as usual is muddled up. Basic values are call by value. Objects are a call by reference.

The swap function C++ version

```
# include <stdio.h>
 void swap(int &, int &); // declaration
 int main()
 {
   int x = 15, y = 42;
   printf("x = %d, y = %d\n", x, y);
   swap(x,y);
   printf("x = %d, y = %d\n", x, y);
 }
 void swap(int &u, int &v) // definition
 {
   int temp;
   temp = u;
   u = v;
   v = temp;
   return;
 }
```

The `void` type

- When the the function does not return any value.
- In certain places where any other type does not make sense.

```
void main(void)
{
  printf("hello world\n");
  return;
}
```

Recursion

Functions can call other functions and even itself

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

The `main` function

- Every statement in a C program has to be part of some function,

The `main` function

- Every statement in a C program has to be part of some function,
- The program execution starts by calling the `main` function,

The `main` function

- Every statement in a C program has to be part of some function,
- The program execution starts by calling the `main` function,
- The return type of `main` can be either `int` or `void`,

The `main` function

- Every statement in a C program has to be part of some function,
- The program execution starts by calling the `main` function,
- The return type of `main` can be either `int` or `void`,
- If return type is `int`, the return value is a way of indicating to the shell if the command has succeeded,

The `main` function

- Every statement in a C program has to be part of some function,
- The program execution starts by calling the `main` function,
- The return type of `main` can be either `int` or `void`,
- If return type is `int`, the return value is a way of indicating to the shell if the command has succeeded,
- It is recommended that you declare `main` with return type `int` and return meaningful status message.