A WORKSHOP ON

THE C PROGRAMMING LANGUAGE

LECTURE 2

Recursion(), see Recursion[1]

A gentle reminder

- Functions are an efficient way to group together lines of code
- Functions are used to perform a certain task
- We can (optionally) pass data into functions which are known as parameters

```
General form of a function definition in c:

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

Classification of functions

Functions can be classified in several ways but the two major ways in which functions differ are on

- return type of the function
- parameters passed to the function

```
Functions based on return type
-  functions that return no value (void)
-  functions that return a value (int, char)
```

Some terms you should be familiar with:

1. **Scope:**
   The scope of an identifier is a part of the program in which the identifier can be used to access its object.
2. **Lifetime:**
   The lifetime of a variable is the interval of time in which storage is bound to the variable.
3. **Visibility:**
   The visibility of an identifier is a region of the program source code from which an identifier's associated object can be legally accessed.

Types of scope

1. **Block:**
   The scope of an identifier with block (or local) scope starts at the declaration point and ends at the end of the block containing the declaration (such block is known as the enclosing block).

   Parameter declarations with a function definition also have block scope, limited to the scope of the function body.

Types of scope

- **File:**
  File scope identifiers, also known as globals, are declared outside of all blocks; their scope is from the point of declaration to the end of the source file.

Types of scope

- **Function:**
  The only identifiers having function scope are statement labels. Label names can be used with goto statements anywhere in the function in which the label is declared.

  Labels are declared implicitly by writing label_name: followed by a statement. Label names must be unique within a function.

Types of scope

- **Function prototype:**
  Identifiers declared within the list of parameter declarations in a function prototype (not as a part of a function definition) have a function prototype scope.

  This scope ends at the end of the function prototype.

Difference between scope and visibility

Scope and visibility usually coincide, though there are circumstances under which an object becomes temporarily hidden by the appearance of a duplicate identifier.

The object still exists but the original identifier cannot be used to access it until the scope of the duplicate identifier ends.

```
void func ( int i ) {
   int j;                     // auto by default
   j = 3;                     // int i and j are in scope and
visible

   {                          // nested block
         double j;               // j is local name in the
nested block
      j = 0.1;                // i and double j are visible;
                            // int j = 3 in scope but hidden
   }
                              // double j out of scope
   j += 1;                    // int j visible and = 4
}
            // i and j are both out of scope
```

Storage class:

- **Auto Storage Class**
  The variables defined using auto storage class are called as local variables. Auto stands for automatic storage class. A variable is in auto storage class by default if it is not explicitly specified.

- **Static Storage Class**
  Static variable is a local variable that retains and stores its value between function calls or block and remains visible only to the function or block in which it is defined.

Recursion:

In computer science, recursion is a method of solving a problem where the solution depends on solutions to smaller instances of the same problem.

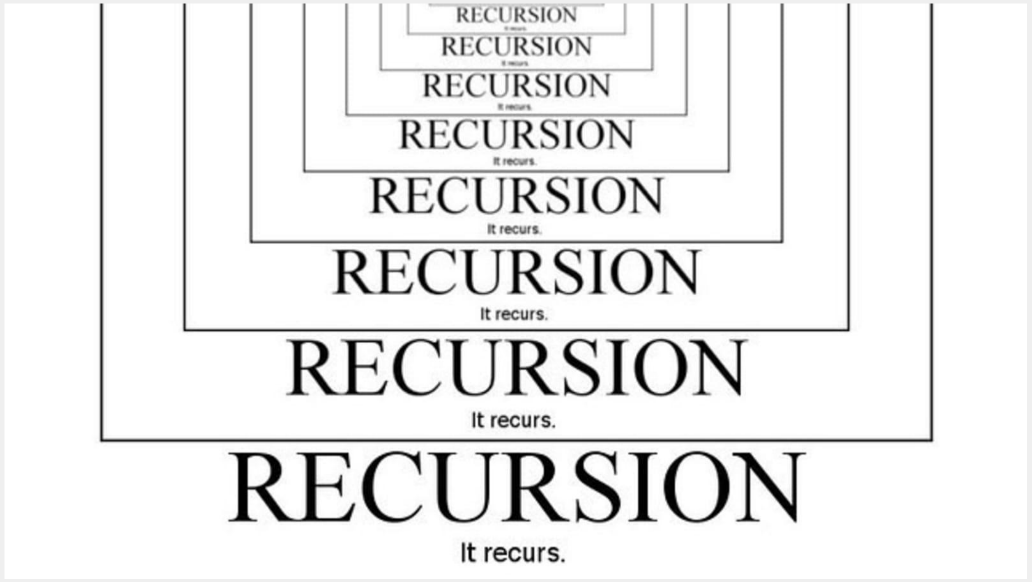
```
A typical recursive function :

return_type function_name(){
   statements;
   function_name();        // Calling itself
}
```

Fibonacci sequence:

A classic example of a recursive procedure is the function used to calculate the nth fibonacci number

F (n) = F (n-1) + F (n-2)



** Need for a base case **