PG – DESD

Module – Embedded C Programming

Trainer - Devendra Dhande

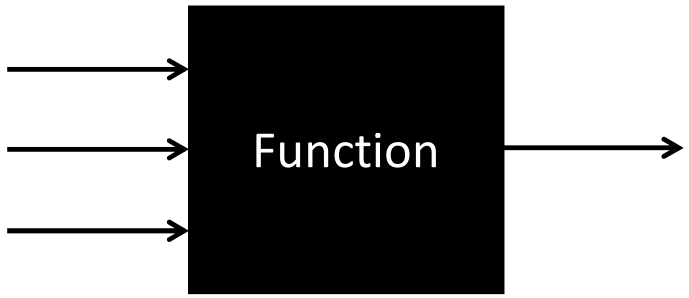
Email – devendra.dhande@sunbeaminfo.com

Mobile No - 9890662093

Functions

- C program is made up of one or more functions.
- C program contains at least one function i.e. main() function.
  - Execution of C program begins from main.
  - It returns exit status to the system.

- Advantages
  - Reusability
  - Readability
  - Maintainability



- Function is set of instructions, that takes zero or more inputs (arguments) and return result (optional).
- Function is a black box.

Functions

- Each function has
  - Declaration

    <return type> <function name> (<list of type of arguments>);
  - Definition

    <return type> <function name> (<list of arguments>)

    {

    //body

    }
  - Call

    <function name>(<list of arguments>)
- A function can be called one or more times.
- Arguments
  - Arguments passed to function → Actual arguments
  - Arguments collected in function → Formal arguments
  - Formal arguments must match with actual arguments

Examples:

1. addition()
2. print_line()
3. factorial()
4. combination()

Functions

- Function Declaration
  - Informs compiler about function name, argument types and return type.
  - Usually written at the beginning of program (source file).
  - Can also be written at start of calling function).
  - Examples:
    - `float divide(int x, int y);`
    - `int fun2(int, int);`
    - `int fun3();`
    - `double fun4(void);`
    - `void fun5(double);`
  - Declaration statements are not executed at runtime.

- Function Definition
  - Implementation of function.
  - Function is set of C statements.
  - It process inputs (arguments) and produce output (return value).

```
float divide(int a, int b) {
        return (float)a/b;
}
```

  - Function can return max one value.
  - Function can be defined in another function.
- Function Call
  - Typically function is called from other function one or more times.

Function execution

- When a function is called, function activation record/stack frame is created on stack of current process.
- When function is completed, function activation record is destroyed.
- Function activation record contains:
  - Local variables
  - Formal arguments
  - Return address
- Upon completion, next instruction after function call continue to execute.

Function types

- User defined functions
  - Declared by programmer
  - Defined by programmer
  - Called by programmer
- Library (pre-defined) functions
  - Declared in standard header files e.g. stdio.h, string.h, math.h, …
  - Defined in standard libraries e.g. libc.so, libm.so, …
  - Called by programmer
- main()
  - Entry point function – code perspective
  - User defined
  - System declared
  - int main(void) {…}
  - int main(int argc, char *argv[]) {…}

Recursion

- Function calling itself is called as recursive function.
- To write recursive function consider
  - Explain process/formula in terms of itself
  - Decide the end/terminating condition
- Examples:
  - $n! = n * (n-1)!$ $\quad\quad$ $0! = 1$
  - $x^y = X * x^{y-1}$ $\quad\quad$ $x^0 = 1$
  - $T_n = T_{n-1} + T_{n-2}$ $\quad\quad$ $T_1 = T_2 = 1$
  - factors(n) = 1^{st} prime factor of n * factors(n)

Recursion execution

```
int fact(int n) {
  int r;
  if(n==0)
    return 1;
  r = n * fact(n-1);
  return r;
}
```

```
int fact(int n) {
  int r;
  if(n==0)
    return 1;
  r = n * fact(n-1);
  return r;
}
```

```
int fact(int n) {
  int r;
  if(n==0)
    return 1;
  r = n * fact(n-1);
  return r;
}
```

```
int fact(int n) {
  int r;
  if(n==0)
    return 1;
  r = n * fact(n-1);
  return r;
}
```

```
int fact(int n) {
  int r;
  if(n==0)
    return 1;
  r = n * fact(n-1);
  return r;
}
```

```
int fact(int n) {
  int r;
  if(n==0)
    return 1;
  r = n * fact(n-1);
  return r;
}
```

```
int main() {
  int res;
  res = fact(5);
  printf("%d", res);
  return 0;
}
```

5! = 5 * 4!

4! = 4 * 3!

3! = 3 * 2!

2! = 2 * 1!

1! = 1 * 0!

0! = 1

Storage class

	Storage	Initial value	Life	Scope
auto / local	Stack	Garbage	Block	Block
register	CPU register	Garbage	Block	Block
static	Data section	Zero	Program	Limited
extern / global	Data section	Zero	Program	Program

- Each running process have following sections:
  - Text
  - Data
  - Heap
  - Stack
- Storage class decides
  - Storage (section)
  - Life (existence)
  - Scope (visibility)
- Accessing variable outside the scope raise compiler error.

Storage class

- Local variables declared inside the function.
  - Created when function is called and destroyed when function is completed.
- Global variables declared outside the function.
  - Available through out the execution of program.
  - Declared using extern keyword, if not declared within scope.
- Static variables are same as global with limited scope.
  - If declared within block, limited to block scope.
  - If declared outside function, limited to file scope.
- Register is similar to local storage class, but stored in CPU register for faster access.
  - register keyword is request to the system, which will be accepted if CPU register is available.

Thank you!

Devendra Dhande <devendra.dhande@sunbeaminfo.com>