Function in C

Function

- Two types of function
  - Library function
    - scanf, printf, gets, puts, getch, sqrt etc.
  - User defined function

Function

```
/*Include header files*/
/*Include function prototypes*/
int main()
{
   return 0;
}
ret-type f1(param-list)
{
   return …;
}
```

Function Prototype

```
#include <stdio.h>
int main()
{
   printf("In main\n");
   myfunc();
   printf("Back in main\n");
   return 0;
}
void myfunc()
{
   printf("In myfunc\n");
}
```

```
'myfunc' undefined; assuming extern returning int
'myfunc' : redefinition; different basic types
```

Function Prototype

- Function prototype declares a function
  - before its use
  - prior to its definition
- Ends with semicolon
- Example:
  - Function:

```
void myfunc()
{
}
```

  - Prototype

```
void myfunc();
```

Function

```
#include <stdio.h>
void myfunc();
int main()
{
    printf("In main\n");
    myfunc();
    printf("Back in main\n");
  return 0;
}
void myfunc()
{
    printf("In myfunc\n");
}
```

Output:

In main

In myfunc

Back in main

- **main** calling function/ caller
- **myfunc** called function
- Control returns to calling function from called function

Function Prototype

- Prototype declares three attributes of a function
  - Its return type
  - Number of parameters
  - Type of parameters
- Compiler need to know the type of data returned by the function
- Default **int** assumed
- Report illegal type conversions
- **main** does not require prototype

Function

- C program contains at least one function
- **main()** must be present exactly once in a program
- No limit on number of functions defined
- There can be function which is defined but not called
- There can be function which is declared but not defined
- One function can not be defined inside other function
- Two or more function can not have same name
- Function and variable name can not be same
- No statements can be written outside of function
- Minimal function is
  - dummy(){}

Function

```
#include <stdio.h>
void one();
void two();
void three();
int main()
{
    printf("In main\n");
    one();
    two();
    three();
    printf("Back in main\n");
    return 0;
}
```

```
void one()
{
    printf("In one\n");
}
void two()
{
    printf("In two\n");
}
void three()
{
    printf("In three\n");
}
```

Output:

```
In main
In one
In two
In three
Back in main
```

Function

- It is possible to call one function from other function
- **main()** can be called from other function

Function

```
#include <stdio.h>
void one();
void two();
void three();
int main()
{
    printf("In main\n");
    one();
    printf("Back in main\n");
    return 0;
}
void three()
{
    printf("In three\n");
}
void two()
{
    printf("In two\n");
    three();
    printf("Back in two\n");
}
void one()
{
    printf("In one\n");
    two();
    printf("Back in one\n");
}
```

Output:

In main
In one
In two
In three
Back in two
Back in one
Back in main

Declaration Vs. Definition

- Declaration: specifies the type of the object
  - Function prototype
- Definition: causes storage for an object to be created
  - Function: which contains the body is definition
  - It is legal to define a function fully before its use
    - Eliminates need of separate prototype

Function

```
#include <stdio.h>
void myfunc()
{
   printf("In myfunc\n");
}
int main()
{
   printf("In main\n");
   myfunc();
   printf("Back in main\n");
   return 0;
}
```

Output:

```
In main
In myfunc
Back in main
```

Function Scope

```
#include <stdio.h>
void myfunc()
{
   printf("i is %d\n", i);
}
int main()
{
   int i=10;
   myfunc();
   return 0;
}
```

Error, **i** is local to **main**

- Local variable cease to exist when the function returns

Function Scope (global variable)

```
#include <stdio.h>
int i;
void myfunc()
{
    printf("i is %d\n", i);
    i=2;
}
int main()
{
    i=10;
    myfunc();
    printf("i is %d\n", i);
    return 0;
}
```

Use of global variable

Function Scope (local variable)

```
#include <stdio.h>
void myfunc()
{
    int i=1;
    printf("i is %d\n", i);
}
int main()
{
    int i=10;
    myfunc();
    printf("i is %d\n", i);
    return 0;
}
```

Output:

```
i is 1
i is 10
```

Return

- If no return type specified: default **int** assumed
- when the **return** statement is encountered: the function returns the control to the caller immediately
- **return** statement can be used without return value
  - return ;
  - Used mostly by **void** functions
- If the return value is not assigned to anything it is lost, but no harm done

Return

```
#include <stdio.h>
void myfunc();
int main()
{
    printf("In main\n");
    myfunc();
    printf("Back in main\n");
    return 0;
}
void myfunc()
{
    printf("In myfunc\n");
    return ;
    printf("In myfunc (will never be printed)\n");
}
```

Output:

```
In main
In myfunc
Back in main
```

Return values

```
#include <stdio.h>
#include <math.h>

int main()
{
   double answer;
   answer=sqrt(161.0);
   printf("%lf\n", answer);
   return 0;
}
```

- **sqrt** is prototyped in math.h

Return values

```
#include <stdio.h>
#include <math.h>

int main()
{
   printf("%lf\n", sqrt(161.0));
   return 0;
}
```

Return values

```
#include <stdio.h>
double sqr5();
int main()
{
   double ans=sqr5();
   printf("%lf\n“, ans);
   return 0;
}
double sqr5()
{
   return 5.0*5.0;
}
```

Output:
25.0

Parameterized Function

```
#include <stdio.h>
double sqr(double);
int main()
{
   double ans=sqr(5.0);
   printf("%lf\n“, ans);
   return 0;
}
double sqr(double x)
{
   return x*x;
}
```

Writing variable name in prototype is not necessary

Output:
25.0

Parameterized Function

```
#include <stdio.h>
void sub(int, int);
int main()
{
    sub(2, 6);
    sub(5, 9);
    return 0;
}

void sub(int x, int y)
{
    printf("%d\n", x-y);
}
```

Writing variable name in prototype is not necessary

Order of argument is important

Parameterized Function

```
#include <stdio.h>
int sub(int, int);
int main()
{
    printf("%d\n", sub(2, 6));
    printf("%d\n", sub(5, 9));
    return 0;
}

int sub(int x, int y)
{
    return x-y;
}
```

Writing variable name in prototype is not necessary

Order of argument is important

Return

- More than one values can not be returned
  - return a, b;

Function Arguments

- To take arguments a function must have special variables
- When **sub** is called is argument is copied in the matching parameter
- *Argument*:
  - The value that is passed to a function
- Known as *formal parameter*
  - The variable that receive the value of the argument inside the function
- Local variables of a function can not have same name as formal parameters

Parameterized Function

```
#include <stdio.h>
void sum(int, int);
int main()
{
    sum(2, 6);
    sum(5, 9);
    return 0;
}
void sum(int x, int y)
{
    int x, y;
    printf("%d\n", x+y);
}
```

Error:
redefinition of formal parameter 'x‘
redefinition of formal parameter ‘y'

Use of library functions

```
#include <stdio.h>
#include <math.h>
#define PI 3.14159265

int main () {
   double x, ret, val;
   x = 60.0;
   val = PI / 180.0;
   ret = sin( x*val );
   printf("The sine of %lf is %lf degrees\n", x, ret);
   return 0;
}
```

Use of library functions

```
#include <stdio.h>
#include <math.h>
#define PI 3.14159265

int main () {
   double x, ret, val;
   x = 60.0;
   val = PI / 180.0;
   ret = cos( x*val );
   printf("The cosine of %lf is %lf degrees\n", x, ret);
   return 0;
}
```

Use of library functions

```
#include <stdio.h>
#include <math.h>
#define PI 3.14159265

int main () {
   double x, ret, val;
   x = 1.0;
   val = 180.0 / PI;
   ret = atan (x) * val;
   printf("The arc tangent of %lf is %lf degrees", x, ret);
   return 0;
}
```

Call by Value

```
#include <stdio.h>
void swap(int x, int y) {
    int t;
    t=x;
    x=y;
    y=t;
}
int main() {
    int a=2, b=5;
    printf("a=%d, b=%d\n", a, b);
    swap(a, b);
    printf("a=%d, b=%d\n", a, b);
    return 0;
}
```

Output:

a=2, b=5

a=2, b=5