
# Introduction to C Programming

Section 6

# Function Input and Output

# Functions

- Every C program starts with main() function

- Functions could be
  - Pre-defined library functions
    - e.g., printf, sin, tan
  - Programmer-defined functions
    - e.g., my_printf, area

# Predefined library functions

- <math.h>
  - Defines common mathematical functions
  - e.g. sin, cos. sqrt, pow
- <stdio.h>
  - Defines core input and output functions
  - e.g. printf, scanf
- <time.h>
  - Defines date and time handling functions
  - e.g. time, clock
- <stdlib.h>
  - Defines pseudo-random numbers generation functions
  - e.g. rand, srand

# C mathematical functions

- double fmod( double x, double y );
  - Computes the remainder of the division operation x/y
- double exp( double arg );
  - Computes the *e* (Euler's number, 2.7182818) raised to the given power arg
- double log( double arg );
  - Computes the natural (base *e*) logarithm of arg
- double log10( double arg );
  - Computes the common (base 10) logarithm of arg
- double sqrt( double arg );
  - Computes square root of arg

# An Example

```
#include <stdio.h>
#include <math.h>
int main(void)
{
    double angle;
    printf("Input angle in radians: \n");
    scanf("%lf", &angle);
    printf("The sine of the angle is %f\n", sin(angle) );
    return 0;
}
```

# Random numbers generation functions

- int rand();
  - Returns a uniformly distributed pseudo-random integral value between 0 and RAND_MAX (0 and RAND_MAX included)
  - RAND_MAX : Expands to an integer constant expression equal to the maximum value returned by the function rand(). This value is implementation dependent.
    - **#define RAND_MAX  32767  /*implementation defined*/**
  - srand() should be called before any calls to rand() to initialize the random number generator
- void srand( unsigned seed );
  - Initializes the built-in random number generator used to generate values for rand() with the seed value seed

# An Example

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

int main(void)
{
  unsigned int seed; /*  Declare variables.  */
  int k;
  /*  Get seed value from the user.  */
  printf("Enter a positive integer seed value: \n");
  scanf("%u",&seed);
  srand(seed);

  /*  Generate and print ten random numbers.  */
  printf("Random Numbers: \n");
  for (k=1; k<=10; k++)
    printf("%i ", rand());
  printf("\n");

  return 0; /*  Exit program.  */
}
```

```
Enter a positive integer seed value:
4
Random Numbers:
51 17945 27159 386 17345 27504 20815 20576 10960 6020
```

# Functions in C

- Queries: Return a value
  - `sin(), fabs()`

- Commands: do some tasks, do not return any value or we don't use the value
  - `printf(...)`
  - `scanf(...)`

# Functions - Definition Structure

- Function 'header'
  - Return data type (if any)
  - Name
    - Descriptive
  - Arguments (or parameter list)
    - Notice: data type and name
- Statements
  - Variable declaration
  - Operations
  - Return value (if any)

```
type function_name (type arg1, type arg2 )

{
    statements;
}
```

# The return command

- `return` finishes running the function
- Function can have multiple return
  - Only one of them runs each time
- The type of the returned value = the result type
  - Otherwise, cast

# Functions that do not return a value

- Use the return type of void
  - void functionName( DataType arg_1,...)
  - void functionName()
  - void functionName( void)

# Function call

- Command function
  - `<function name> (inputs);`
- Query function
  - `<variable> = <function name> (inputs);`
  - Inputs should match by function definition
  - Functions are called by another function
  - Function call comes inside in a function

# Example

```
void my_info(void){
  printf("My name is Ahmad Ahmadi\n");
  printf("My student number: 95222222\n");
}

int main(void){
  my_info();
  printf("---------------\n");
  my_info();
  return 0;
}
```

# Function Call – An Example

- If the function returns a value, then the returned value need to be assigned to a variable so that it can be stored

```
int GetUserInput (void); /* function prototype*/
int main(void)
{
    int input;
    input = GetUserInput( );
    return(0); /* return 0; */
}
```

- However, it is perfectly okay (syntax wise) to just call the function without assigning it to any variable if we want to ignore the returned value

- We can also call a function inside another function

```
printf("User input is: %d", GetUserInput( ));
```

# An Example

- If the function requires some arguments to be passed along, then the arguments need to be listed in the bracket ( ) according to **the specified order**

```
void Calc(int  ..., double ..., char..., int...){...}

int main(void)
{
    int a, b;
    double c;
    char d;
    ...
    Calc(a, c, d, b);
     return (0);
}
```

Function Call

# Purposes of Function Prototype

- So compiler knows how to compile calls to that function, i.e.,
  - number and types of arguments
  - type of result
- As part of a “contract” between developer and programmer who uses the function
- As part of hiding details of *how* it works and exposing *what* it does.
- A function serves as a “black box.”
- Prototype only needed if function definition comes after use in program

# Scopes vs. Blocks

- Scopes are determined by Blocks
  - Start with { and finished by }
  - Example: statements of a function, statement of a
  - **`if`** or **`while`**, …
- Variables
  - Can be declared in a block
  - Can be used in the declared block
  - Cannot be used outside the declared block
- The Declared block is the scope of the variable

# Nested Scopes/Blocks

- Scopes can be nested
- Example: Nested **`if`**, nested **`for`**, …

```
int main(){ //block 1  int i;
  { //block 2  int j;
     { //block 3  int k;
     }
     int m;
  }
  return 0;
}
```

# Variables in Nested Blocks

- All variables from outer block can be used inner blocks
  - Scope of outer block contains the inner block
- Variables in inner block cannot be used in outer block
  - Scope of the inner block does not contains the outer block

# Local Variables

- All defined variables in a function are the local variable of the function
- Can ONLY be used in the function, not other functions

```
void func(void){  int
i,j;  float f;
/* These are local variables */
}
int main(void){
    i = 10;  /* compile error. why?
    f = 0;   /* compile error. why?
}
```

# Call by value

- In call by value mechanism
  - The values are copied to the function
  - Only the copy of variable's value (copy of actual parameter's value) is passed to the function.
- If we change values in the function
  - The copied version is changed
  - The original value does not affected

**Any modification to the passed value inside the function will not affect the actual value**

# Call by reference

- Call by **reference**
  - In this method, the reference (memory address) of the variable is passed to the function. **Any modification passed done to the variable inside the function will affect the actual value**

# Call by reference

```
#include <stdio.h>
void CalByVal(a, b)
{
    a = 0;  b = 10;
}
void CalByRef(int *a, int *b) // CalByRef(int *p, int *q)
{
    *a = 0;  *b = -5;
}
int main(void)
{
   int a = 1, b = 5;
   printf("Before cal CalByVal: a = %d, b = %d\n", a, b);
   CalByVal(a, b);
   printf("After cal CalByVal: a = %d, b = %d\n", a, b);

   printf("Before cal CalByRef: a = %d, b = %d\n", a, b);
   CalByRef(&a, &b);
   printf("After cal CalByRef: a = %d, b = %d\n", a, b);
   return 0; /*  Exit program.  */
}
```

# Call by reference

- Instead of `product()`
  - `prod_sum()`
  - How can I get the function to give both product *and* sum?
    - put * in front of variable name in prototype and function definition
    - put & in front of variable names in function call

```
#include <stdio.h>
void prod_sum(double x, double y,
                  double *ptr1, double *ptr2);
int main()
{
  double var1 = 3.0, var2 = 5.0;
  double prod, sum;
  prod_sum(var1, var2, &prod, &sum);

   printf("var1= %g\n"
          "var2= %g\n",var1, var2);
  printf("prod= %g\n" "sum= %g\n", prod, sum);
}
/* function definition */
void prod_sum(double A, double B,
  double *rslt_prod, double *rslt_sum)
 {
   *rslt_prod = A * B;
   *rslt_sum = A + B;
 }
```