

Function Subprograms (Functions)

- A **function** is a set of instructions that can be used again and again, each time on a possibly different set of values (parameters).
- Once a function is specified, it is used by **calling** it and **passing** it the **actual parameter values** on which to perform its task.

- **Library (Standard) Functions:**

```
/* program that calls a library function sqrt */
#include <stdio.h>
#include <math.h>
void main ()
{
    double w,y;

    w = 16.0;
    y = sqrt(w);    //w is the actual parameter or argument)
    printf("%f\n",y);
}
```

- **The Function sqrt():**

```
/* function to find the square root of a parameter x */
double sqrt(double x)    //x is a formal (or dummy) parameter
{
    instructions to compute the square root of x
    return the value computed
}
```

- **General Form of a Function:**

```
return_data_type function_name(formal_parameter_list)
{
    statements of the function;
}
```

- **Programmer Defined Functions:**

```
/* function to return three times
 * the input parameter 'numb'
 */
int triple(int numb)
{
    return (3 * numb);
}
```

- **Using Function triple:**

```
/* program to test function triple */
#include <stdio.h>
int triple(int);                // function prototype
void main()
{
    int a,b,c,numb;

    a = 5;
    b = triple(a);
    printf("original value is %d, its triple is %d\n",a,b);

    numb = 2;
    c = triple(numb);
    printf("original value is %d, its triple is %d\n",numb,c);

    numb = triple(a-1);
    printf("original value is %d, its triple is %d\n",a-1,numb);

    c = triple(4);
    printf("original value is %d, its triple is %d\n",4,c);
}
```

Function Prototypes (or Declarations):

- The function prototype gives the compiler the information it needs to perform **type checking** when it finds a call to the function.
- The prototype contains:
 - the data type of the return value,
 - the name of the function,
 - the number and data type of parameters
 - e.g.,
 int triple(int);
or
 int triple(int numb); //the variable name is a dummy
or
 int triple(int x);
- The prototype must appear before a call to the function.
- When using a programmer defined function, the programmer must include the function prototype explicitly.
- When using a library function, the programmer typically inserts a **header file** that includes the function prototypes for the library functions.
- **Header files** are files ending in .h (e.g., stdio.h). They are included into the program through the **#include** directive. the directive tells the preprocessor to include the given file at this point in the program

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

Local Variables in Functions

- **Local Variables:**

```
/* function to find the largest of three integers a,b,c */
int max3(int a, int b, int c)
{
    int maxsofar;          //maxsofar is a local variable

    /* find the larger of a and b */
    if (a >= b)
        maxsofar = a;
    else
        maxsofar = b;
    /* now compare maxsofar to c */
    if (maxsofar < c)
        maxsofar = c;
    return (maxsofar);
}
```

- **Using the Function max3:**

```
/* program to test max3 */
#include <stdio.h>
int max3(int,int,int);
void main()
{
    int x,y,z,ans;

    x = 3;
    y = 5;
    z = 7;
    ans = max3(x,y,z);
    printf("%d\n",ans);
}
```

- **Using Multiple return Statements:**

```
/* function to find the largest of three integers a,b,c */
int max3(int a, int b, int c)
{
    int maxsofar;          //maxsofar is a local variable

    /* find the larger of a and b */
    if (a >= b)
        maxsofar = a;
    else
        maxsofar = b;
    /* now compare maxsofar to c */
    if (maxsofar < c)
        return (c);
    else
        return (maxsofar);
}
```

- **A More Complex Call to max3:**

```
/* program to test max3 */
#include <stdio.h>
#include <math.h>
int max3(int,int,int);
void main()
{
    int p,q;

    p = 10;
    q = -7;
    printf("%d\n",max3(p+ 2, 12, abs(q) * 3));
}
```

Location of Functions

- In C, functions are typically included within the same module as the main program by placing them after the compiler directives (#include, etc.) and function prototypes. The code for all functions is typically placed either before the main program or after the main program. As an alternative, the functions can be compiled separately and later **linked** to the main program when it is compiled.

- **Complete Program Using a Function Subprogram:**

```
/* find the sum of the first 'number_to_sum' squares
 * using a function.
 */
#include <stdio.h>
int sumofsquares(int);           //function prototype

/* function to find sum of the squares */
int sumofsquares(int n)
{
    int i, sum = 0;

    for (i = 1; i <= n; i++ )
        sum += (i * i);
    return (sum);
}

void main()
{
    int number_to_sum;

    printf("Enter the number of squares to be summed: ");
    scanf("%d",&number_to_sum);
    printf("%d is the sum of the first %d squares.\n",
        sumofsquares(number_to_sum),number_to_sum);
}
```

- **Alternate Structure:**

```
/* find the sum of the first 'number_to_sum' squares
 * using a function.
 */
#include <stdio.h>
int sumofsquares(int);           //function prototype

void main()
{
    int number_to_sum;

    printf("Enter the number of squares to be summed: ");
    scanf("%d",&number_to_sum);
    printf("%d is the sum of the first %d squares.\n",
        sumofsquares(number_to_sum),number_to_sum);
}

/* function to find sum of the squares */
int sumofsquares(int n)
{
    int i, sum = 0;

    for (i = 1; i <= n; i++ )
        sum += (i * i);
    return (sum);
}
```

void Functions

- Functions that do not return any value are called **void functions**.

- Example:

```
/* program with function to find and print both the larger
 * and smaller of two values
 */
#include <stdio.h>
void print_max_min(int,int);      //function prototype

void print_max_min(int x, int y)
{
    if (x > y)
        printf("%d is the larger and %d is the smaller\n",x,y);
    else
        printf("%d is the larger and %d is the smaller\n",y,x);
    return;
}

void main()
{
    int a,b;

    a = 5;
    b = 3;
    print_max_min(a,b);
}
```

Parameterless Functions

- You can have functions without parameters.

- Example:

```
/* write a function to print heading for a table */  
#include < stdio.h>  
void printheadings(void);           //or void printheadings();  
  
void printheadings(void)           //or void printheadings()  
{  
    printf("\n\t\tSales data for the Past 15 Years\n\n");  
    printf("\tYear\t\tSales\t\tExpenses\t\tProfits\n");  
    return;  
}  
  
void main()  
{  
    printheadings();  
}
```

Input-Process-Output (I-P-O) Comments

- In describing any function, including the main program, there are three important parts:
 - The **Input**
what it expects from the outside world,
 - The **Process**
what it accomplishes (and possibly how),
 - The **Output**
what it returns and outputs to the outside world.

- Example

```
/*
 * Function sumofsquares
 * Input:
 *   n - the number of squares to sum
 * Process:
 *   finds the sum of the first n squares
 *   by finding 1* 1 + 2* 2 + ... n* n
 * Output:
 *   returns the sum of the first n squares
 */
int sumofsquares(int n)
{
    int i, sum = 0;

    for (i = 1; i <= n; i+ + )
        sum + = (i * i);
    return (sum);
}
```

Using Functions to Produce a Multiplication Table

- **Pseudocode for Multiplication Table:**

*print the headings at the top of the page
for each multiplicand (m1) from 1 to 10
 print a line of output showing m1 times each multiplier
 (m2) from 1 to 10.*

- **Pseudocode Refinement:**

*printheadings();
for each multiplicand (m1) from 1 to 10
 print a line of output showing m1 times each multiplier
 (m2) from 1 to 10.*

- **Further Refinement:**

*printheadings();
for each multiplicand (m1) from 1 to 10
 printrow (m1);*

- **More Refinement:**

*printheadings();
for (m1 = 1; m1 <= 10; m1+ +)
 printrow (m1);*

Program to Produce a Multiplication Table

```
/* multiplication table for the integers 1 to 10 */
#include <stdio.h>
void printheadings(void);      //function prototypes
void printrow (int);

void main()
{
    int m1;                    //m1= multiplicand

    printheadings();
    for (m1 = 1; m1 <= 10; m1+ + )
        printrow (m1);
}

/* Function printheadings
 * Input:
 *     none
 * Process:
 *     prints headings for a multiplication chart
 * Output:
 *     prints the table headings
 */
void printheadings(void)
{
    int m2;

    printf("This is a Multiplication Table from 1 to 10\n\n");
    printf(" X");
    /* loop to print the heading of multipliers */
    for (m2 = 1; m2 <= 10; m2+ + )
        printf("%5d",m2);
    printf("\n");
    return;
}
```

```

/* Function printrow
 * Input:
 *   m1 - the current multiplicand
 * Process:
 *   prints a row of a multiplication table by calculating the
 *   first 10 multiples of the multiplicand.
 * Output:
 *   prints a row of the table
 */
void printrow(int m1)
{
    int m2;                //multiplier

    printf("%2d",m1);      //prints the multiplicand
    for (m2 = 1; m2 <= 10; m2+ + )
        printf("%5d", m1 * m2); //prints the row
    printf("\n");
    return;
}

```

- **Return value of the Function printf():**
printf() returns the number of characters printed, unless an error occurred, in which case it returns a negative value.

- Example:

```
int val,count;
```

```
scanf("%d",&val);  
count = printf("The value read in is %d\n",val);  
printf("The first call to printf printed %d characters\n",count);
```

- Typical Output

```
The value read in is 89
```

```
The first call to printf printed 24 characters
```

- **Return value of the Function scanf():**

scanf() returns the number of successful conversions, (this can be 0). If an end of the input stream is reached, the return value is the constant EOF (EOF is defined in stdio.h).

- Example:

```
int val,num;
```

```
num = scanf("%d",&val);  
while (num == 1) {  
    printf("The value read in is %d\n",val);  
    num = scanf("%d",&val);  
}
```

```
while (scanf("%d",&val) != EOF) //terminates upon EOF  
    printf("The value read in is %d\n",val);
```

- This first loop will terminate if either an EOF is reached or the user entered a value which could not be converted to an integer (e.g., a character). Note: To create an EOF input, in Windows press CTRL-Z. (In UNIX it is CTRL-D)

Parameters that are Pointers

- **Problem:**

Write a C function that finds the max and min of two integers and returns both values to the calling program.

- **The Function:**

```
/* Function find_max_min
 * Input:
 *   x - first integer
 *   y - second integer
 *   max - memory address where to put larger value
 *   min - memory address where to put smaller value
 * Process:
 *   The function stores the larger of x and y into * max and
 *   the smaller of x and y into * min
 * Output:
 *   * max - the larger of the values
 *   * min - the smaller of the values
 */
void find_max_min(int x, int y, int * max, int * min)
{
    if (x > y) {
        * max = x;
        * min = y;
    }
    else {
        * max = y;
        * min = x;
    }
    return;
}
```

- **Notes:**

- **max and min are pointers** to memory locations that contain integers. (This is indicated by the **int *** data type.)
- * max (* min) refers to the contents of the memory location pointed to by max (min).
- A pointer parameter expects an **address** to be passed to it.

- **Driver Program:**

```
/* ... */
#include <stdio.h>
void find_max_min(int, int, int *, int * );

void main()
{
    int a= 1, b= 2;
    int larger,smaller;

    find_max_min(a, b, &larger, &smaller);
    printf("The larger is %d and the smaller is %d\n",
        larger,smaller);
}
```

- **Notes:**

- The **& (ampersand) operator** means 'address of'.
- &larger (&smaller) refers to the address of the memory location of the variable called larger (smaller).