

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 <iostream>
#include <cmath>
using namespace std;
int main ()
{
    double w,y;

    w = 16.0;
    y = sqrt(w);    //w is the actual parameter (or argument)
    cout << y << endl;
    return 0;
}
```

- **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 <iostream>
using namespace std;
```

```
int triple(int); // function prototype
```

```
int main ()
{
    int a,b,c,numb;

    a = 5;
    b = triple(a);
    cout << "original value is " << a << " , its triple is "
         << b << endl;

    numb = 2;
    c = triple(numb);
    cout << "original value is " << numb << " , its triple is "
         << c << endl;

    numb = triple(a-1);
    cout << "original value is " << a-1 << " , its triple is "
         << numb << endl;

    c = triple(4);
    cout << "original value is " << 4 << " , its triple is "
         << c << endl;

    return 0;
}
```

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 text files. 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 <iostream>  
#include <cmath>
```

Note: In C, header files had to have an extension of .h
In C + + , this is not necessary.

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 <iostream>
using namespace std;

int max3(int,int,int);          //function prototype

int main ()
{
    int x,y,z,ans;

    x = 3;
    y = 5;
    z = 7;
    ans = max3(x,y,z);
    cout << ans << endl;
    return 0;
}
```

- **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 <iostream>
#include <cmath>
using namespace std;

int max3(int,int,int);          //function prototype

int main()
{
    int p,q;

    p = 10;
    q = -7;
    cout << max3(p + 2, 12, abs(q) * 3) << endl;
    return 0;
}
```

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 <iostream>
using namespace std;

int sumofsquares(int);           //function prototype

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

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

int main()
{
    int number_to_sum;

    cout << "Enter the number of squares to be summed: ";
    cin >> number_to_sum;
    cout << sumofsquares(number_to_sum) <<
        " is the sum of the first " << number_to_sum <<
        "squares." << endl;
    return 0;
}
```

● **Alternate Structure:**

```
/* find the sum of the first 'number_to_sum' squares
 * using a function.
 */
#include <iostream>
using namespace std;

int sumofsquares(int);           //function prototype

int main()
{
    int number_to_sum;

    cout << "Enter the number of squares to be summed: ";
    cin >> number_to_sum;
    cout << sumofsquares(number_to_sum) <<
         " is the sum of the first " << number_to_sum <<
         "squares." << endl;
    return 0;
}

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

    for (int 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 <iostream>
using namespace std;
```

```
void print_max_min(int,int);          //function prototype
```

```
void print_max_min(int x, int y)
```

```
{
    if (x > y)
        cout << x << " is the larger and "
             << y << " is the smaller" << endl;
    else
        cout << y << " is the larger and "
             << x << " is the smaller" << endl;
    return;
}
```

```
int main()
{
    int a,b;

    a = 5;
    b = 3;
    print_max_min(a,b);
    return 0;
}
```

Parameterless Functions

- You can have functions without parameters.

- Example:

```
/* write a function to print heading for a table */  
#include <iostream>  
using namespace std;
```

```
void printheadings(void);           //or void printheadings();
```

```
void printheadings(void)           //or void printheadings()  
{  
    cout << "\n\t\tSales data for the Past 15 Years" << endl  
        << endl;  
    cout << "\tYear\t\tSales\t\tExpenses\t\tProfits" << endl;  
    return;  
}
```

```
int main()  
{  
    printheadings();  
    return 0;  
}
```

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 sum = 0;  
  
    for (int 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 (int m1 = 1; m1 <= 10; m1++)
 printrow(m1);*

Program to Produce a Multiplication Table

```
/* multiplication table for the integers 1 to 10 */
#include <iostream>
using namespace std;

void printheadings(void);          //function prototypes
void printrow(int);

int main()
{
    printheadings();
    for (int m1 = 1; m1 <= 10; m1++)
        printrow(m1);           //m1 = multiplicand
    return 0;
}

/* Function printheadings()
 * Input:
 *     none
 * Process:
 *     prints headings for a multiplication chart
 * Output:
 *     prints the table headings
 */
void printheadings(void)
{
    cout << "\tThis is a Multiplication Table from 1 to 10"
         << endl << endl;
    cout.width(5);
    cout << "X";
    /* loop to print the heading of multipliers */
    for (int m2 = 1; m2 <= 10; m2++)
    {
        cout.width(5);
        cout << m2;
    }
    cout << endl;
    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)
{
    cout.width(5);
    cout << m1;           //prints the multiplicand
    for (int m2 = 1; m2 <= 10; m2++) //m2 = multiplier
    {
        cout.width(5);
        cout << m1 * m2; //prints the product
    }
    cout << endl;
    return;
}

```

Reference Parameters

- **Program:**

```
/* program to try to add one to a function parameter */
#include <iostream>
using namespace std;

void trytoadd1(int);           //function prototype

int main()
{
    int k = 5;

    cout << "in main - before call: " << k << endl;
    trytoadd1(k);
    cout << "in main - after call: " << k << endl;
    return 0;
}

/* function that tries to add one to its parameter */
void trytoadd1(int x)        //x is receiving the value of k
{
    cout << "in function - before adding: " << x << endl;
    x + +;
    cout << "in function - after adding: " << x << endl;
    return;
}
```

- Output:

```
in main - before call: 5
in function - before adding: 5
in function - after adding: 6
in main - after call: 5
```

WHAT WENT WRONG???

- **Correct Version of Program:**

```
/* program to add one to function parameter */
#include <iostream>
using namespace std;

void add1(int &);           //function prototype

int main()
{
    int k = 5;

    cout << "in main - before call: " << k << endl;
    add1(k);
    cout << "in main - after call: " << k << endl;
    return 0;
}

/* function that adds one to its parameter */
void add1(int &x)          //x is receiving a reference to k
{
    cout << "in function - before adding: " << x << endl;
    x++;
    cout << "in function - after adding: " << x << endl;
    return;
}
```

- Output:

```
in main - before call: 5
in function - before adding: 5
in function - after adding: 6
in main - after call: 6
```

- Notes:

- **int &** means **reference to an integer**. This means that the formal parameter and the actual parameter (or argument) are the same, in the sense that any change to the value of the formal parameter will cause a like change to the value of the actual parameter. (The formal parameter becomes an **alias** for the actual parameter.)

Returning More than One Value from a Function

- **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 - reference where to put larger value
 *   min - reference 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;
}
```

● **Driver Program:**

```
/* ... */
#include <iostream>
using namespace std;

void find_max_min(int, int, int &, int &); //function prototype

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

    find_max_min(a, b, larger, smaller);
    cout << "The larger is " << larger
         << " and the smaller is " << smaller << endl;

    find_max_min(5, a + b, larger, smaller);
    cout << "Now the larger is " << larger
         << " and the smaller is " << smaller << endl;

    return 0;
}
```

Program to Illustrate File I/O within a Fuction

```
/* multiplication table for the integers 1 to 10 */
#include <iostream>
#include <fstream>
using namespace std;

//function prototypes
void printheadings(ofstream &);
void printrow(ofstream &, int);

int main()
{
    // declare and open the output file
    // ofstream outfile("output.txt"); //comment-out for debugging
    ofstream outfile("con"); //un-comment for debugging

    printheadings(outfile);
    for (int m1 = 1; m1 <= 10; m1 + +)
        printrow(outfile, m1); //m1 = multiplicand

    outfile.close(); //close the output file

    return 0;
}
```

```

/* Function printheadings()
 * Input:
 *   out1 - a reference to the output file
 * Process:
 *   prints headings for a multiplication chart
 * Output:
 *   prints the table headings
 */
void printheadings(ofstream &out1)
{
    out1 << "\tThis is a Multiplication Table from 1 to 10"
        << endl << endl;
    out1.width(5);
    out1 << "X";
    /* loop to print the heading of multipliers */
    for (int m2 = 1; m2 <= 10; m2 + +)
    {
        out1.width(5);
        out1 << m2;
    }
    out1 << endl;
    return;
}

```

```

/* Function printrow()
 * Input:
 *   out2 - a reference to the output file
 *   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(ofstream &out2, int m1)
{
    out2.width(5);
    out2 << m1;           //prints the multiplicand
    for (int m2 = 1; m2 <= 10; m2++) //m2 = multiplier
    {
        out2.width(5);
        out2 << m1 * m2; //prints the product
    }
    out2 << endl;
    return;
}

```