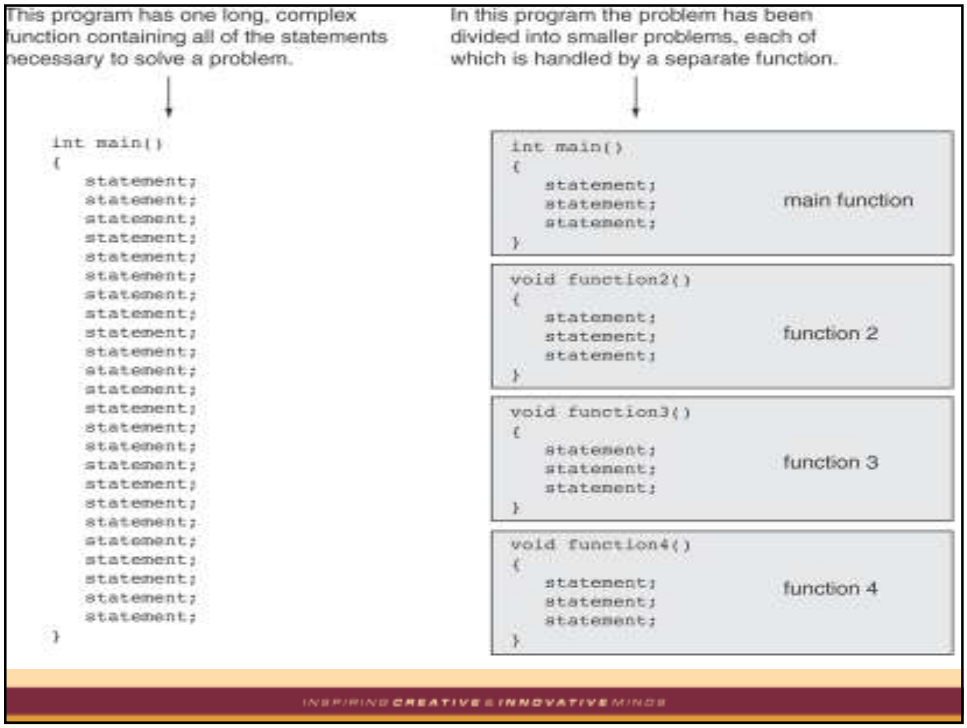
UTM
RESEARCH UNIVERSITY

Introduction to Function

UTM

Modular Programming

- Modular programming: breaking a program up into smaller, manageable functions or modules
- Function: a collection of statements to perform a task
- Motivation for modular programming:
  - Improves maintainability of programs
  - Simplifies the process of writing programs



UTM

Function

- A collection of statements that performs a specific task.
- Commonly used to break a problem down into small manageable pieces.
- In C++, there are 2 types of function:
  - Library functions
  - User-defined functions

Library Functions

- "Built-in" functions that come with the compiler.
- The source code (definition) for library functions does NOT appear in your program.
- To use a library function, you simply need to include the proper header file and know the name of the function that you wish to use.
  - **#include** *compiler directive*

Library Functions (cont.)

- **Libraries under discussion at this time:**

Compiler directive	Purpose
<ctype>	Character classification and conversion
<cmath>	Math functions
<stdlib>	Data conversion
<time>	Time functions

Library Functions: Math Functions

- Required header: #include <cmath>
- Example functions

Function	Purpose
abs(x)	returns the absolute value of an integer.
pow(x,y)	calculates x to the power of y. If x is negative, y must be an integer. If x is zero, y must be a positive integer.
pow10(x)	calculates 10 to the power of x.
sqrt(x)	calculates the positive square root of x. (x is >=0)

Library Functions: Example 1

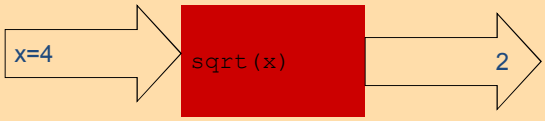
```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
    double area, radius;

cout<< "This program calculates the area of a circle.\n";
   cout<<"What is the radius of the circle? ";
   cin>>radius;
   area=3.14159 * pow(radius,2.0);
   cout<<"The area is " << area <<endl;
   system ("pause");
return 0;
}
```

Library Function: Example 2

```
#include <iostream>
#include <cmath>
using namespace std;
main(){
    int nom1,nom2, result;
    cout<<"Enter two numbers";
    cin>>nom1>>nom2;
    if ((nom1<0)||(nom2<0))
    {   cout<<"negative number/s";
    }
    else
    {
       result= sqrt(nom1 + nom2);
       cout<<"The square root of "<< nom1+nom2 << "is"
    << result;}
}
```

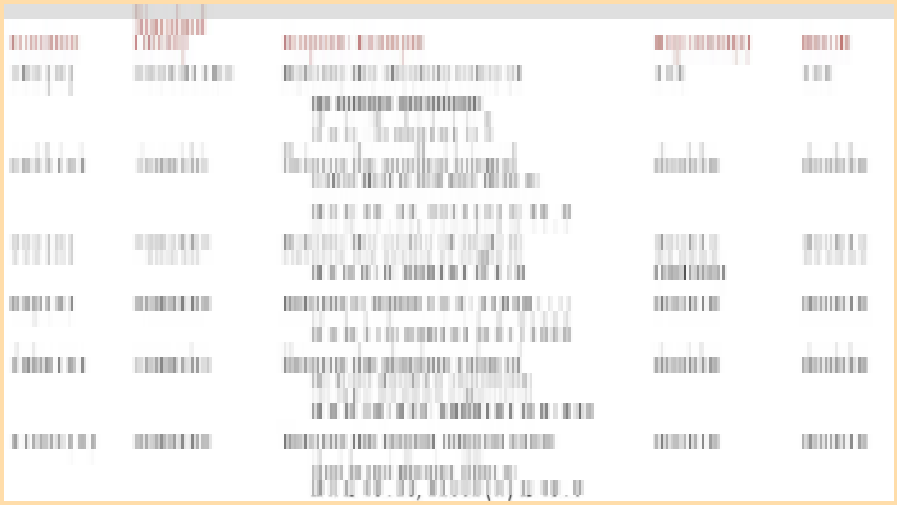
Library Functions (cont.)

- A collection of specialized functions.
- C++ promotes code reuse with predefined classes and functions in the standard library
- The functions work as a black box:



Library Functions (cont.)

- Some Mathematical Library Functions



Library Functions (cont.)

- Some Mathematical Library Functions

INSPIRING CREATIVE & INNOVATIVE MINDS

More Mathematical Library Functions

- Require `cmath` header file
- Take `double` as input, return a `double`
- Commonly used functions:

`sin`	Sine
`cos`	Cosine
`tan`	Tangent
`sqrt`	Square root
`log`	Natural (e) log
`abs`	Absolute value (takes and returns an int)

More Mathematical Library Functions

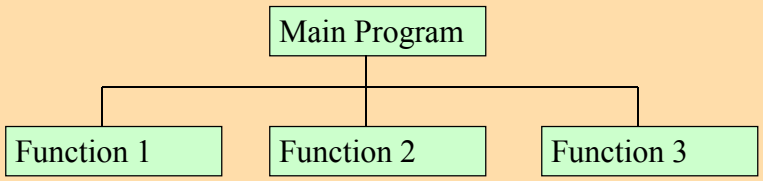
- These require `cstdlib` header file
- `rand()`: returns a random number (`int`) between `0` and the largest int the compute holds. Yields same sequence of numbers each time program is run.
- `srand(x)`: initializes random number generator with `unsigned int x`

In-Class Exercise

- Do Lab 10, Exercise 2, No. 1 (pg. 141)

User-Defined Functions

- User-defined functions are created by you, the programmer.
- Commonly used to break a problem down into small manageable pieces.
- You are already familiar with the one function that every C++ program possesses: **int main()**
  - Ideally, your **main( )** function should be very short and should consist primarily of function calls.



UTM

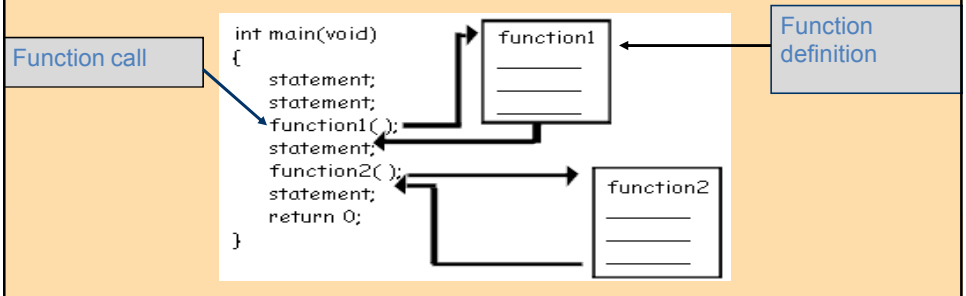
Defining and Calling Functions

- Function call: statement causes a function to execute
- Function definition: statements that make up a function

UTM

User-Defined Functions (cont.)

- Every functions must have:
  - Function call: statement causes a function to execute
  - Function definition: statements that make up a function



Function Definition

- Definition includes:
  - return type: data type of the value that function returns to the part of the program that calls it
  - name: name of the function. Function names follow same rules as variables
  - parameter list: variables containing values passed to the function
  - body: statements that perform the function's task, enclosed in { }

User-Defined Functions : Function Definition (cont.)

- *The general form of a function definition in C++ is as follows:*

```
function-returntype function-name( parameter-list )
{
local-definitions;
function-implementation;
}
```

UTM

Function Definition

Note: The line that reads `int main()` is the function header.

UTM

Function Return Type

- If a function returns a value, the type of the value must be indicated:

```
int main()
```

- If a function does not return a value, its return type is `void`:

```
void printHeading()
{
      cout << "Monthly Sales\n";
}
```

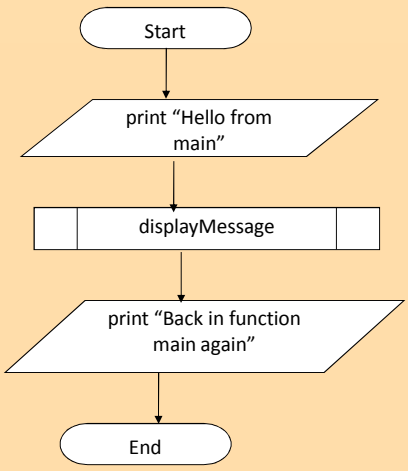
Calling a Function

- To call a function, use the function name followed by `()` and `;`

```
printHeading();
```

- When called, program executes the body of the called function
- After the function terminates, execution resumes in the calling function at point of call.

The flowchart

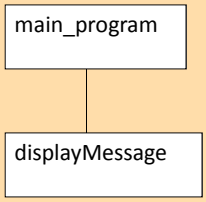


The pseudo code

- Start
- Print “Hello from main”
- call displayMessage
- Print “Back in function main again”
- End

- displayMessage:
  - Print “Hello from the function displayMessage”

The Structure Chart



Calling a Function - example

Program 6-1

```
1  // This program has two functions: main and displayMessage
2  #include <iostream>
3  using namespace std;
4
5  //****************************************
6  // Definition of function displayMessage  *
7  // This function displays a greeting.     *
8  //****************************************
9
10 void displayMessage()
11 {
12    cout << "Hello from the function displayMessage.\n";
13 }
14
15 //****************************************
16 // Function main                          *
17 //****************************************
18
19 int main()
20 {
21    cout << "Hello from main.\n";
22    displayMessage();
23    cout << "Back in function main again.\n";
24    return 0;
25 }
```

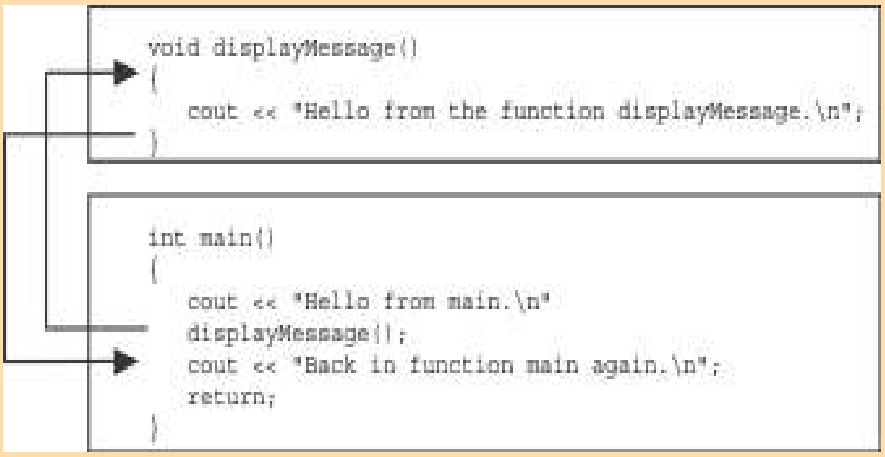
Function definition

Function call

Program Output

```
Hello from main.
Hello from the function displayMessage.
Back in function main again.
```

Flow of Control in Program 6-1



User-Defined Functions: Example 2

```
1.  #include <iostream>
2.  #include <cmath>
3.  using namespace std;

4.  float distance(float x, float y)
5.  {    float dist;
6.       dist = sqrt(x * x + y * y);
7.       return dist;
8.   }

9.  void main()
10. {
11.     float x,y,dist;
12. cout << "Testing function distance(x,y)"  << endl;
13. cout << "Enter values for x and y: ";
14. cin >> x >> y;
15. dist = distance(x,y);
16. cout << "Distance of (" << x << ',' << y << ") from origin
    is " << dist << endl << "Tested" << endl;
17. }
```


Calling Functions

- `main` can call any number of functions
- Functions can call other functions
- Compiler must know the following about a function before it is called:
  - name
  - return type
  - number of parameters
  - data type of each parameter

UTM

In-Class Exercise

- Do Lab 11, Exercise 1, No 1 (pg. 147-149)
- Which of the following function headers are valid? If they are invalid, explain why.
  - `one (int a, int b)`
  - `int thisone(char x)`
  - `char another (int a, b)`
  - `double yetanother`

UTM

Function Prototypes

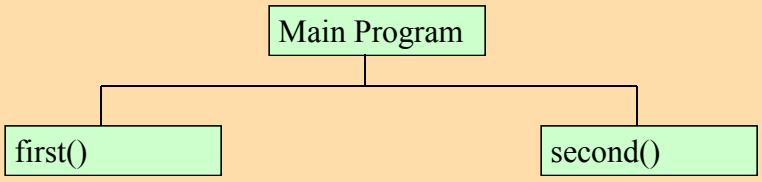
- Ways to notify the compiler about a function before a call to the function:
  - Place function definition before calling function's definition
  - Use a function prototype (function declaration) – like the function definition without the body
    - Header: `void printHeading()`
    - Prototype: `void printHeading();`

User-Defined Functions: Function Prototypes

```
#include <iostream>

void first();
void second();

void main()
{
    cout<< "Starting in main function \n";
   first();
   second();
   cout<<"Control back to main\n";
}

void first()
{   cout<<"Inside the first function\n";}

void second()
{   cout<<"Inside the second function\n";}
```

Function prototypes

Structured Chart



UTM

Prototype Notes

- Place prototypes near top of program
- Program must include either prototype or full function definition before any call to the function – compiler error otherwise
- When using prototypes, can place function definitions in any order in source file

UTM

User-Defined Functions: Functions with No Parameters

```
1.  #include <iostream>
2.  void printhi();

3.  void main(){
4.  cout << "Testing function printhi()" << endl;
    printhi();
5.  cout << "Tested" << endl;
6.  } // End of main

7.  // Function Definitions
8.  void printhi()
9.  {    cout << "Hi \n";  }
```

Sending Data into a Function

- Can pass values into a function at time of call:

```
c = pow(a, b);
```

- Values passed to function are arguments
- Variables in a function that hold the values passed as arguments are parameters

A Function with a Parameter Variable

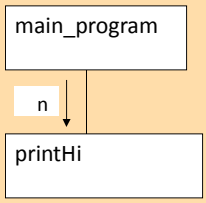
```
void displayValue(int num)
{
   cout << "The value is " << num << endl;
}
```

The integer variable `num` is a parameter.
It accepts any integer value passed to the function.

User-Defined Functions: Functions with Parameters and No Return Values (1)

```
#include <iostream>
using namespace std;
void printhi(int);

void main(){
    int n;
    cout <<"Enter a value for n: ";
    cin >> n;
    printhi(n);
    cout << "Tested \n";}

void printhi(int n)
{    int i;
    for (i = 0; i < n; i++)
       cout << "Hi \n";
}
```


The Structure Chart



User-Defined Functions: Functions with Parameters and No Return Values (2)

```
#include <iostream>
using namespace std;

void displayValue(int);

void main()
{
    cout<<"Passing number 5 to displayValue\n";
   displayValue(5);
   cout<<"Back in main\n";
}

void displayValue(int n)
{
    cout<<"The value is " << n << endl;
}
```


Other Parameter Terminology

- A parameter can also be called a formal parameter or a formal argument
- An argument can also be called an actual parameter or an actual argument

Parameters, Prototypes, and Function Headers

- For each function argument,
  - the prototype must include the data type of each parameter inside its parentheses
  - the header must include a declaration for each parameter in its ()

```
void evenOrOdd(int);   //prototype
void evenOrOdd(int num) //header
evenOrOdd(val);        //call
```


Function Call Notes

- Value of argument is copied into parameter when the function is called
- A parameter's scope is the function which uses it
- Function can have multiple parameters
- There must be a data type listed in the prototype () and an argument declaration in the function header () for each parameter
- Arguments will be promoted/demoted as necessary to match parameters

User-Defined Functions: Passing Multiple Arguments

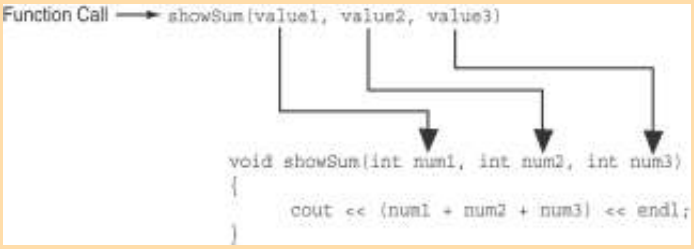
When calling a function and passing multiple arguments:

- the number of arguments in the call must match the prototype and definition
- the first argument will be used to initialize the first parameter, the second argument to initialize the second parameter, etc.

User-Defined Functions: Passing Multiple Arguments (cont.)

```
#include <iostream>
using namespace std;
void showSum(int, int, int);

int main()
{
    int value1, value2, value3;

   cout<<"Enter 3 integers: ";
   cin>> value1 >> value2 >> value3;
   showSum(value1, value2, value3);

   return 0;
}

void showSum(int a, int b, int c)
{
    cout<<"The sum: "<<a+b+c;
}
```

Passing Multiple Arguments (cont..)



The function call in line 18 passes value1, value2, and value3 as a arguments to the function.

In-Class Exercise

- What is the output of this program?

```
#include <iostream>

// Function prototype
void showDouble(int);

int main(){
  int num;
  for (num = 0; num < 10; num++)
        showDouble(num);
   system("pause");
   return 0;
}

//Definition of function
void showDouble(int value) {
 cout<<value <<"\t";
 cout << (value * 2)<< endl;
}
```

In-Class Exercise

- What is the output of this program?

```
#include <iostream>

// Function prototype
void func1(double, int);

int main(){
 int x = 0; double y = 1.5;
 cout << x << " " <<y<< endl;
 func1 (y, x);
 cout << x << " " <<y<< endl;
 system ("pause");
 return 0;
}

void func1(double a, int b){
 cout << a << " " <<b<< endl;
 a=0.0; b=10;
 cout << a << " " <<b<< endl;
}
```


In-Class Exercise

- Do Lab 11, Exercise 1, No. 1 - 6 (pg. 148 – 149)
- Do Lab 11, Exercise 3, No. 1 (pg. 163)
- Do Lab 11, Exercise 3, No. 3