# User-Defined Functions: Passing Data

- Passing by Value
- Passing by Reference

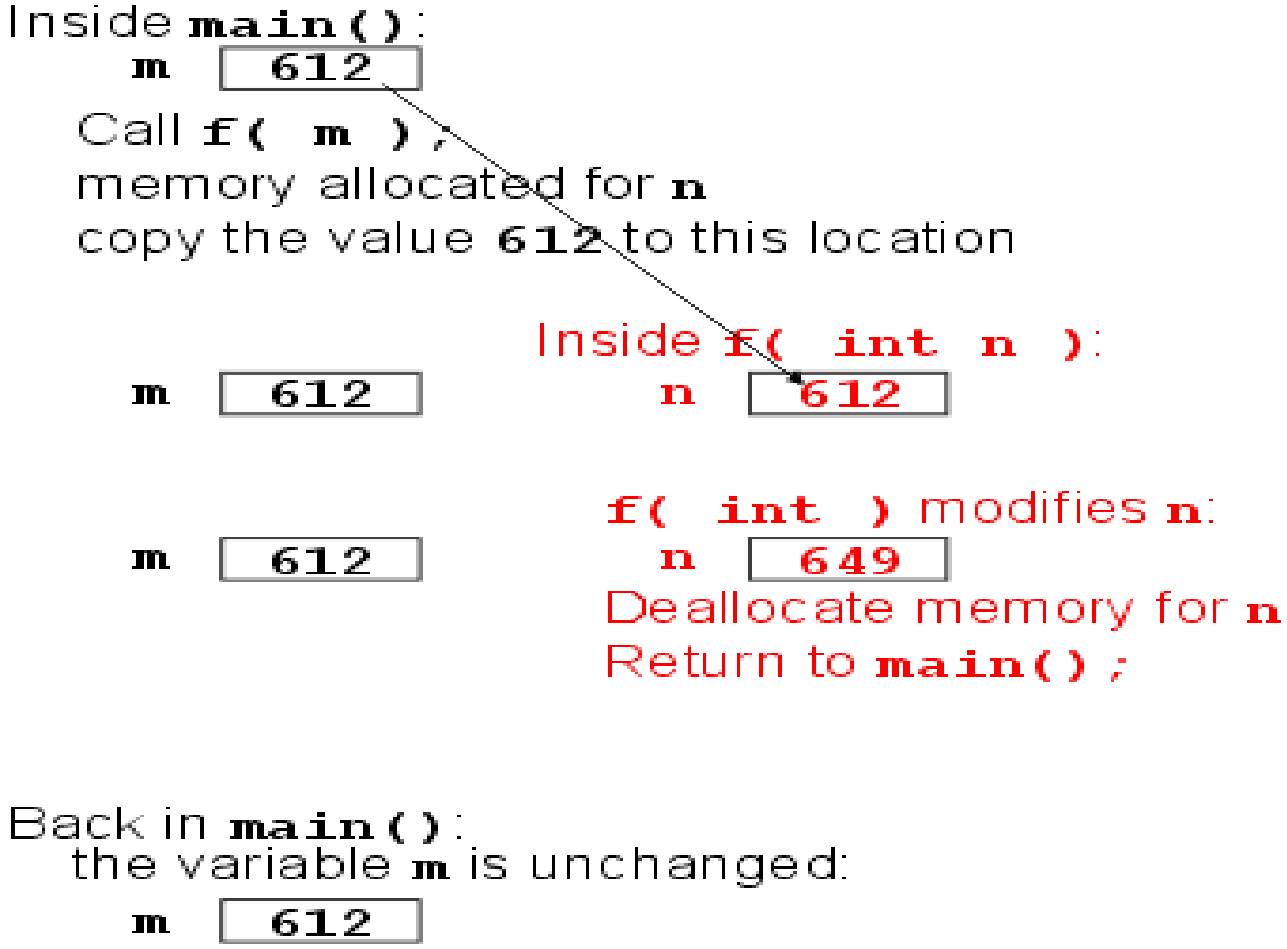
# Passing Data by Value

- Pass by value: when an argument is passed to a function, its value is **copied** into the parameter.
- Changes to the parameter in the function do not affect the value of the argument

# User-Defined Functions: Passing Data by Value (cont.)

```
01  #include <iostream>
02  using namespace std;
03
04  void f( int n ) {
05     cout << "Inside f( int ), the value of the parameter is "
       << n << endl;
06     n += 37;
07     cout << "Inside f( int ), the modified parameter is now "
       << n << endl;}
08
09  int main() {
10     int m = 612;
11
12     cout << "The integer m = " << m << endl;
13     cout << "Calling f( m )..." << endl;
14     f( m );
15     cout << "The integer m = " << m << endl;
16     return 0;
17  }
```

# User-Defined Functions: Passing Data by Value (cont.)



# Passing Information to Parameters by Value

- Example: `int val=5;`

```
evenOrOdd(val);
```


- `evenOrOdd` can change variable `num`, but it will have no effect on variable `val`

# Using Functions in Menu-Driven Programs

- Functions can be used
  - to implement user choices from menu
  - to implement general-purpose tasks:
    - Higher-level functions can call general-purpose functions, minimizing the total number of functions and speeding program development time

# The `return` Statement

- Used to end execution of a function
- Can be placed anywhere in a function
  - Statements that follow the `return` statement will not be executed
- Can be used to prevent abnormal termination of program
- In a `void` function without a `return` statement, the function ends at its last }

# Returning a Value from a Function

- A function can return a value back to the statement that called the function.
- You've already seen the `pow` function, which returns a value:

```
double x;
x = pow(2.0, 10.0);
```

# Returning a Value from a Function

- In a value-returning function, the `return` statement can be used to return a value from function to the point of call. Example:

```
int sum(int num1, int num2)
{
  double result;
  result = num1 + num2;
  return result;
}
```

# A Value-Returning Function

Return Type

```
int sum(int num1, int num2)
{
    double result;
    result = num1 + num2;
    return result;
}
```

Value Being Returned

# A Value-Returning Function

```
int sum(int num1, int num2)
{
    return num1 + num2;
}
```

Functions can return the values of expressions, such as `num1 + num2`

# The `return` Statement - Example

**Program 6-11**

```
1  // This program uses a function to perform division. If division
2  // by zero is detected, the function returns.
3  #include <iostream>
4  using namespace std;
5
6  // Function prototype.
7  void divide(double, double);
8
9  int main()
10 {
11    double num1, num2;
12
13    cout << "Enter two numbers and I will divide the first\n";
14    cout << "number by the second number: ";
15    cin >> num1 >> num2;
16    divide(num1, num2);
17    return 0;
18 }
```

# The `return` Statement - Example

## *Program 6-11(Continued)*

```
20  //*************************************************************
21  // Definition of function divide.                             *
22  // Uses two parameters: arg1 and arg2. The function divides arg1*
23  // by arg2 and shows the result. If arg2 is zero, however, the  *
24  // function returns.                                          *
25  //*************************************************************
26
27  void divide(double arg1, double arg2)
28  {
29     if (arg2 == 0.0)
30     {
31        cout << "Sorry, I cannot divide by zero.\n";
32        return;
33     }
34     cout << "The quotient is " << (arg1 / arg2) << endl;
35  }
```

Return to called function

**Program Output with Example Input Shown in Bold**

```
Enter two numbers and I will divide the first
number by the second number: 12 0 [Enter]
Sorry, I cannot divide by zero.
```

# Returning a Value From a Function

**Program 6-12**

```
1  // This program uses a function that returns a value.
2  #include <iostream>
3  using namespace std;
4
5  // Function prototype
6  int sum(int, int);
7
8  int main()
9  {
10    int value1 = 20,    // The first value
11        value2 = 40,    // The second value
12        total;          // To hold the total
13
14    // Call the sum function, passing the contents of
15    // value1 and value2 as arguments. Assign the return
16    // value to the total variable.
17    total = sum(value1, value2);
18
19    // Display the sum of the values.
20    cout << "The sum of " << value1 << " and "
21         << value2 << " is " << total << endl;
22    return 0;
23 }
```

# Returning a Value From a Function

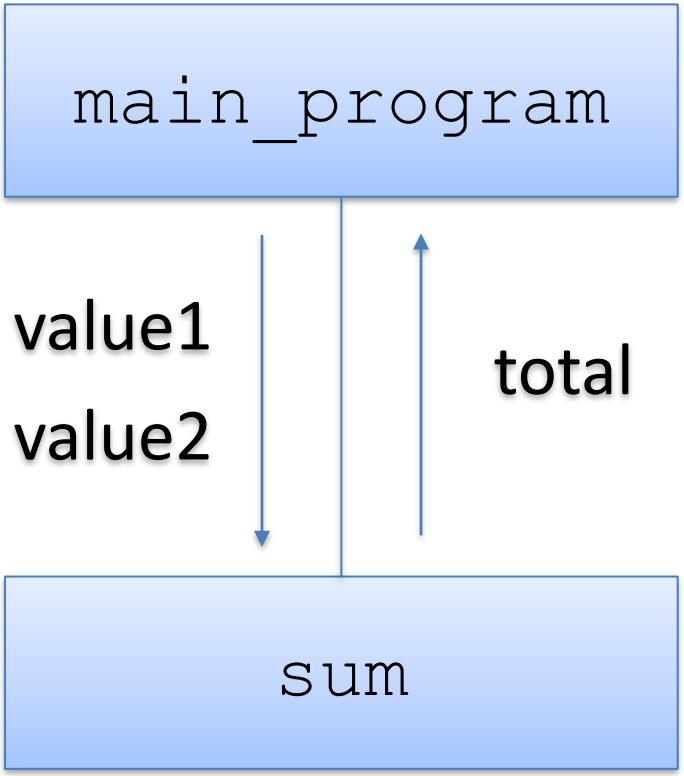
## *Program 6-12 (Continued)*

```
24
25  //*************************************************
26  // Definition of function sum. This function returns  *
27  // the sum of its two parameters.                     *
28  //*************************************************
29
30  int sum(int num1, int num2)
31  {
32     return num1 + num2;
33  }
```

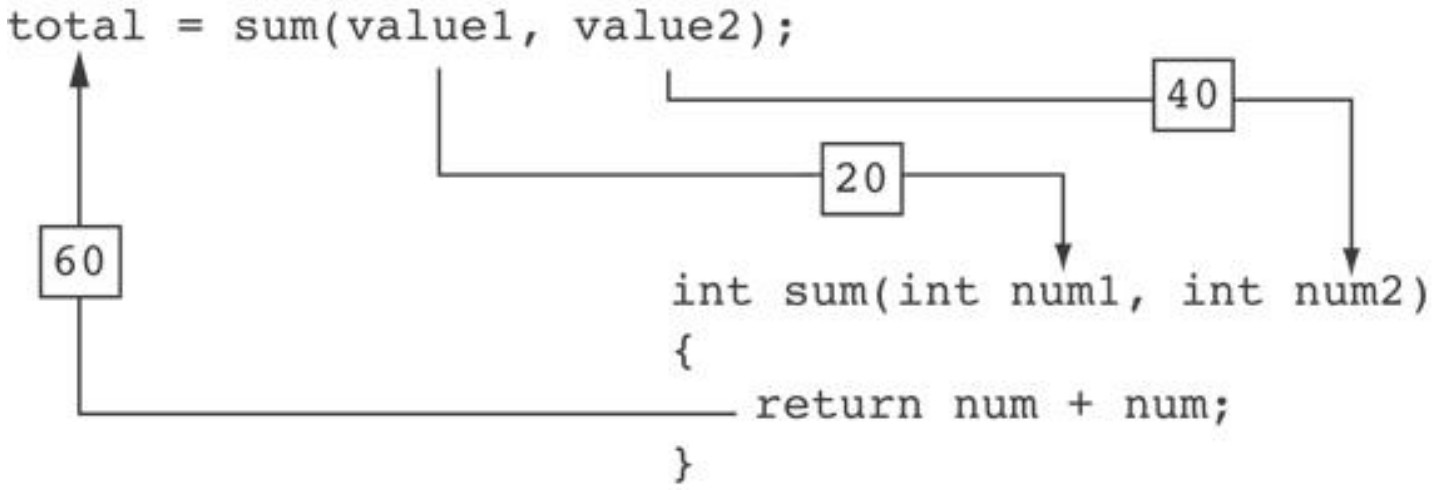
**Program Output**

```
The sum of 20 and 40 is 60
```

# The Structure Chart



# Returning a Value From a Function




The statement in line 17 calls the sum function, passing `value1` and `value2` as arguments. The return value is assigned to the `total` variable.

# Returning a Value From a Function

- The prototype and the definition must indicate the data type of return value (not `void`)

- Calling function should use return value:
  - assign it to a variable
  - send it to `cout`
  - use it in an expression

# Returning a Boolean Value

- Function can return `true` or `false`
- Declare return type in function prototype and heading as `bool`
- Function body must contain `return` statement(s) that return `true` or `false`
- Calling function can use return value in a relational expression

# Returning a Boolean Value

**Program 6-14**

```
1  // This program uses a function that returns true or false.
2  #include <iostream>
3  using namespace std;
4
5  // Function prototype
6  bool isEven(int);
7
8  int main()
9  {
10    int val;
11
12    // Get a number from the user.
13    cout << "Enter an integer and I will tell you ";
14    cout << "if it is even or odd: ";
15    cin >> val;
16
```

# Returning a Boolean Value

**Program 6-14** *(continued)*

```
17      // Indicate whether it is even or odd.
18      if (isEven(val))
19         cout << val << " is even.\n";
20      else
21         cout << val << " is odd.\n";
22      return 0;
23   }
24
25   //*************************************************************
26   // Definition of function isEven. This function accepts an    *
27   // integer argument and tests it to be even or odd. The function  *
28   // returns true if the argument is even or false if the argument  *
29   // is odd. The return value is an bool.                       *
30   //*************************************************************
31
32   bool isEven(int number)
33   {
34      bool status;
35
36      if (number % 2)
37         status = false;    // number is odd if there's a remainder.
38      else
39         status = true;     // Otherwise, the number is even.
40      return status;
41   }
```

**Program Output with Example Input Shown in Bold**

```
Enter an integer and I will tell you if it is even or odd: 5 [Enter]
5 is odd.
```

# In-Class Exercise

```
#include <iostream>
using namespace std;
void try1(int p);
int try3(int r);

int main()
{ int a=2;
 cout << a <<endl;
 try1(a);
 cout << a <<endl;
 int b=3;
 cout << b <<endl;
 int c=4;
 try3(c);
 cout << c <<endl;
 c=try3(c);
 cout << c <<endl;
 cout << try3(5) <<endl;
  return 0;}
```

```
void try1(int p)
{
 p++;
 cout << p <<endl;
}

int try3(int r)
{
  return r*r;
}
```

# In-Class Exercise

- Write a function prototype and header for a function named `distance`. The function should return a `double` and have a two `double` parameters: `rate` and `time`.
- Write a function prototype and header for a function named `days`. The function should return an integer and have three integer parameters: `years`, `months` and `weeks`.
- Examine the following function header, then write an example call to the function.
  - `void showValue(int quantity)`

# In-Class Exercise

- The following statement calls a function named `half`. The `half` function returns a value that is half that of the argument. Write the function.

```
result = half(number);
```

- A program contains the following function:

```
int cube (int num)
{
    return num*num*num;
}
```

Write a statement that passes the value 4 to this function and assigns its return value to the variable `result`.

# In-Class Exercise

- Write a C++ program to calculate a rectangle's area. The program consists of the following functions:
  - `getLength` – This function should ask the user to enter the rectangle's length, and then returns that value as a double.
  - `getWidth` – This function should ask the user to enter the rectangle's width, and then returns that value as a double.
  - `getArea` – This function should accept the rectangle's length and width as arguments and return the rectangle's area.
  - `displayData` – This function should accept the rectangle's length, width and area as arguments, and display them in an appropriate message on the screen.
  - `main` – This function consists of calls to the above functions.

# Local and Global Variables

- Variables defined inside a function are *local* to that function. They are hidden from the statements in other functions, which normally cannot access them.
- Because the variables defined in a function are hidden, other functions may have separate, distinct variables with the same name.

# Local and Global Variables - Example

**Program 6-15**

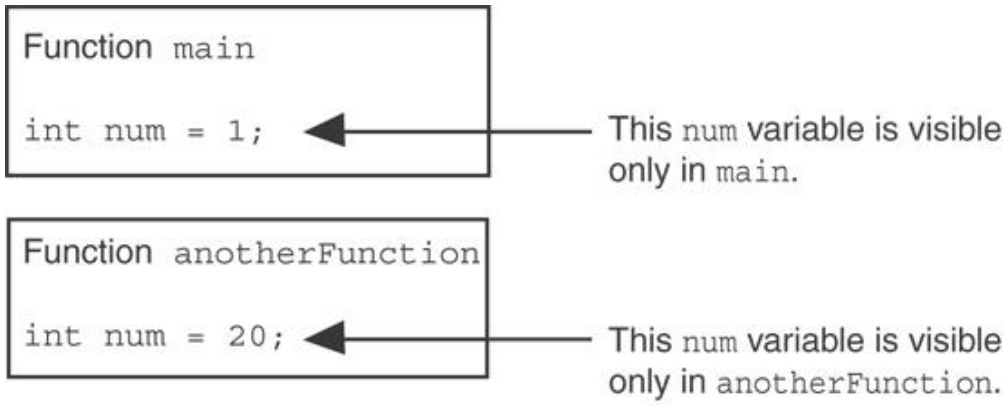
```
1  // This program shows that variables defined in a function
2  // are hidden from other functions.
3  #include <iostream>
4  using namespace std;
5
6  void anotherFunction(); // Function prototype
7
8  int main()
9  {
10    int num = 1;    // Local variable
11
12    cout << "In main, num is " << num << endl;
13    anotherFunction();
14    cout << "Back in main, num is " << num << endl;
15    return 0;
16 }
17
18 //*************************************************
19 // Definition of anotherFunction                  *
20 // It has a local variable, num, whose initial value  *
21 // is displayed.                                  *
22 //*************************************************
23
24 void anotherFunction()
25 {
26    int num = 20;  // Local variable
27
28    cout << "In anotherFunction, num is " << num << endl;
29 }
```

**Program Output**

```
In main, num is 1
In anotherFunction, num is 20
Back in main, num is 1
```

# Local and Global Variables - Example

- When the program is executing in `main`, the `num` variable defined in `main` is visible.
- When `anotherFunction` is called, however, only variables defined inside it are visible, so the `num` variable in `main` is hidden.



# Local Variable Lifetime

- A function's local variables exist only while the **function is executing**. This is known as the *lifetime* of a local variable.

- When the function begins, its local variables and its parameter variables are **created** in memory, and when the function ends, the local variables and parameter variables are **destroyed**.

- This means that any value stored in a local variable is lost between calls to the function in which the variable is declared.

# Global Variables and Global Constants

- A **global** variable is any variable defined outside all the functions in a program.
- The scope of a global variable is the portion of the program from the variable definition to the end.
- This means that a global variable can be accessed by ***all*** functions that are defined after the global variable is defined

# Global Variables and Global Constants

- You should **avoid** using global variables because they make programs difficult to debug.

- Any global that you create should be ***global constants***.

# Global Constants – Example

**Program 6-18**

```
1  // This program calculates gross pay.
2  #include <iostream>
3  #include <iomanip>
4  using namespace std;
5
6  // Global constants
7  const double PAY_RATE = 22.55;     // Hourly pay rate
8  const double BASE_HOURS = 40.0;    // Max non-overtime hours
9  const double OT_MULTIPLIER = 1.5; // Overtime multiplier
10
11 // Function prototypes
12 double getBasePay(double);
13 double getOvertimePay(double);
14
15 int main()
16 {
17    double hours,              // Hours worked
18           basePay,            // Base pay
19           overtime = 0.0,     // Overtime pay
20           totalPay;           // Total pay
```

Global constants defined for values that do not change throughout the program's execution.

# Global Constants – Example

The constants are then used for those values throughout the program.

```
29      // Get overtime pay, if any.
30      if (hours > BASE_HOURS)
31         overtime = getOvertimePay(hours);
```

```
56      // Determine base pay.
57      if (hoursWorked > BASE_HOURS)
58         basePay = BASE_HOURS * PAY_RATE;
59      else
60         basePay = hoursWorked * PAY_RATE;
```

```
75      // Determine overtime pay.
76      if (hoursWorked > BASE_HOURS)
77      {
78         overtimePay = (hoursWorked - BASE_HOURS) *
79                  PAY_RATE * OT_MULTIPLIER;
```

# Initializing Local and Global Variables

- Local variables are not automatically initialized. They must be initialized by programmer.

- Global variables (not constants) are automatically initialized to `0` (numeric) or `NULL` (character) when the variable is defined.

# Static Local Variables

- Local variables only exist while the function is executing. When the function terminates, the contents of local variables are lost.
- `static` local variables retain their contents between function calls.
- `static` local variables are defined and initialized only the first time the function is executed. `0` is the default initialization value.

# Local Variables - Example

**Program 6-20**

```
1  // This program shows that local variables do not retain
2  // their values between function calls.
3  #include <iostream>
4  using namespace std;
5
6  // Function prototype
7  void showLocal();
8
9  int main()
10 {
11    showLocal();
12    showLocal();
13    return 0;
14 }
15
```

# Local Variables - Example

**Program 6-20** *(continued)*

```
16  //*********************************************************
17  // Definition of function showLocal.                      *
18  // The initial value of localNum, which is 5, is displayed. *
19  // The value of localNum is then changed to 99 before the   *
20  // function returns.                                      *
21  //*********************************************************
22
23  void showLocal()
24  {
25     int localNum = 5; // Local variable
26
27     cout << "localNum is " << localNum << endl;
28     localNum = 99;
29  }
```

**Program Output**

```
localNum is 5
localNum is 5
```

In this program, each time `showLocal` is called, the `localNum` variable is re-created and initialized with the value 5.

# A Different Approach, Using a Static Variable

**Program 6-21**

```
1  // This program uses a static local variable.
2  #include <iostream>
3  using namespace std;
4
5  void showStatic(); // Function prototype
6
7  int main()
8  {
9     // Call the showStatic function five times.
10    for (int count = 0; count < 5; count++)
11       showStatic();
12    return 0;
13 }
14
```

# Using a Static Variable - Example

**Program 6-21** *(continued)*

```
15  //*************************************************************
16  // Definition of function showStatic.                         *
17  // statNum is a static local variable. Its value is displayed *
18  // and then incremented just before the function returns.     *
19  //*************************************************************
20
21  void showStatic()
22  {
23     static int statNum;
24
25     cout << "statNum is " << statNum << endl;
26     statNum++;
27  }
```

**Program Output**

```
statNum is 0
statNum is 1
statNum is 2
statNum is 3
statNum is 4
```

`statNum` is automatically initialized to 0. Notice that it retains its value between function calls.

# Using a Static Variable - Example

If you do initialize a local static variable, the initialization only happens once. See Program 6-22…

**Program 6-22** *(continued)*

```
16  //*************************************************************
17  // Definition of function showStatic.                         *
18  // statNum is a static local variable. Its value is displayed *
19  // and then incremented just before the function returns.     *
20  //*************************************************************
21
22  void showStatic()
23  {
24     static int statNum = 5;
25
26     cout << "statNum is " << statNum << endl;
27     statNum++;
28  }
```

**Program Output**

```
statNum is 5
statNum is 6
statNum is 7
statNum is 8
statNum is 9
```

# In-Class Exercise

- Given the following programs compare the output and reason the output.

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

void showVar();

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

void showVar() {
  static int var = 10;
   cout << var << endl;
   var++;
}
```

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

void showVar();

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

void showVar() {
  int var = 10;
  cout << var << endl;
  var++;
}
```

# In-Class Exercise

- Identify global variables & local variables in the following program. What is the output?

```
#include <iostream>
using namespace std;
int j = 8;

int main()
{
    int i=0;
    cout<<"i: "<<i<<endl;
    cout<<"j: "<<j<<endl;
    system("pause");
    return 0;
}
```

- Identify global variables, local variables and static local variables in the following program. What is the output?

```
#include <iostream>
using namespace std;
int j = 40;

void p()
{     int i=5;
      static int j=5;
      i++;
      j++;
      cout<<"i: "<<i<<endl;
      cout<<"j: "<<j<<endl;
}
int main()
{    p();
     p();
     return 0;}
```