National University of computer and emerging sciences

Foundation for Advancement of Science and Technology

Programming Fundamentals

Function in C++

BS (SE) Fall-2025

Tasks

- Write a function isEven(int n) that returns true if n is even, otherwise false.
- Write a function power(int base, int exponent) that returns the result of base^exponent.
- Write a function reverseNumber(int n) that returns the reverse of a number.
- Create a function countVowels(string s) that counts the number of vowels in a string.

Agenda

- Function overview
- Modular programming
- Defining and calling functions
- Function prototype (declaration)

Function overview

- Function is a set of instruction that are designed to perform a specific task
  - Function is a complete and independent program
  - Divide a large program into smaller units
  - User defined and built-in functions
- Modular programming:
  - breaking a program up into smaller, manageable functions or modules
- Some advantages of functions
  - Functions makes a program clear and understandable
  - Finding errors will be easily (easy to debug)
  - Avoids code repetition and saves development time (code reusability)
  - Makes a program modification easy without changing the structure of a program (easy code maintenance)
  - Improves maintainability of programs
  - Simplifies the process of writing programs

Example

This program has one long, complex function containing all of the statements necessary to solve a problem.

```
int main()
{
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
}
```

In this program the problem has been divided into smaller problems, each of which is handled by a separate function.

```
int main()
{
   statement;
   statement;
   statement;
}
```

main function

```
void function2()
{
   statement;
   statement;
   statement;
}
```

function 2

```
void function3()
{
   statement;
   statement;
   statement;
}
```

function 3

```
void function4()
{
   statement;
   statement;
   statement;
}
```

function 4

User define functions

- Functions created by user as part of the program
- These functions are used for a specific use/purpose
- User defined function has three parts
  - Function declaration
  - Function definition
  - Function calling

Defining and Calling Functions

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

Function Definition

- Definition includes: (Declarator and Body of Function)
  - return type: data type of the value that function returns to the part of the program that called 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 { }

Example program

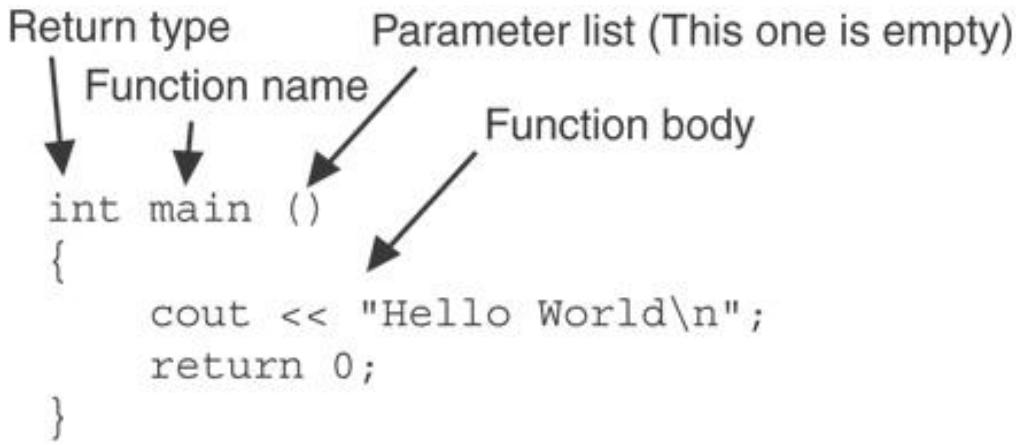
```
#include<iostream>
using namespace std;
int sum(int, int);
int main()
{

    int s;
    s = sum(5, 10);
    cout << "The sum is = " << s;
    return 0;
}

int sum(int a, int b)
{
    int sum = 0;
    sum = a + b;
    return sum;
}
```

Output:
The sum is = 15

Function Definition



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

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.

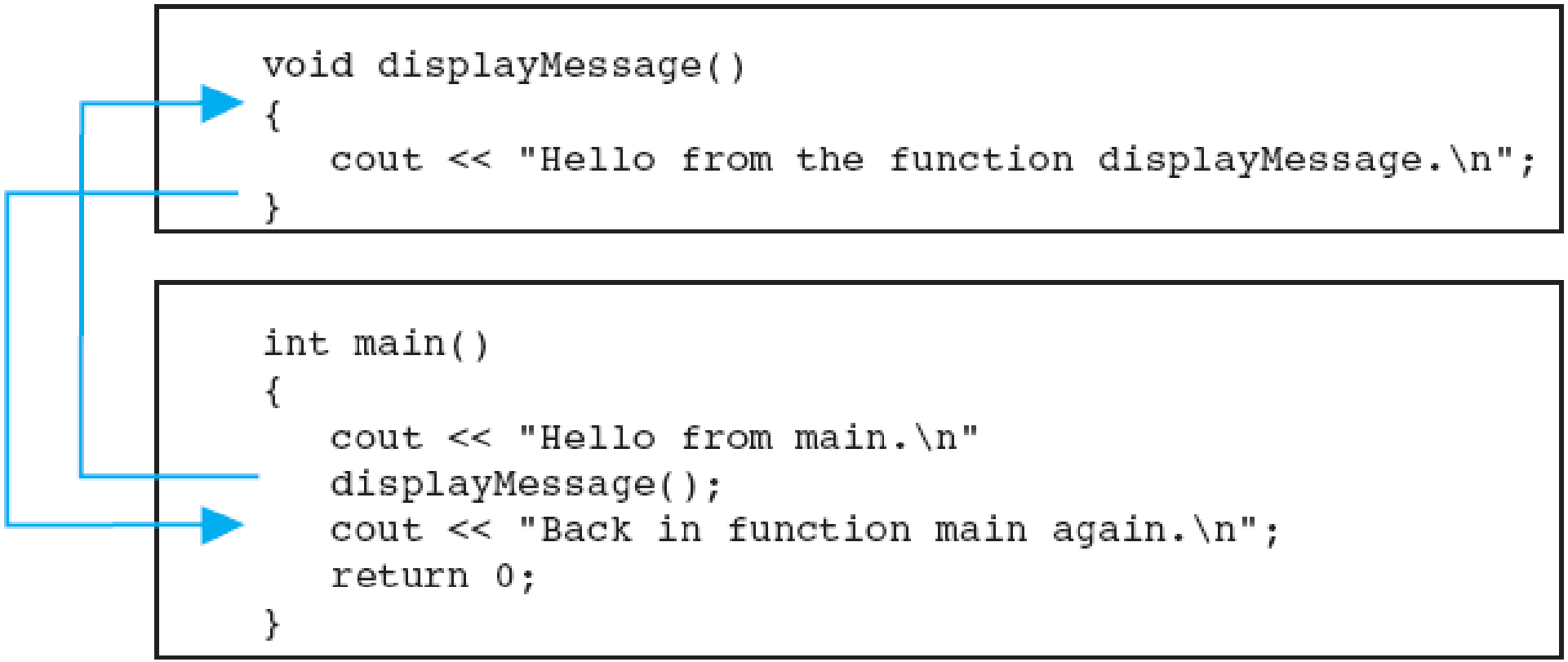
Example Program

```
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 }
```

Program Output

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

Flow of Control in Program 6-1



```
void displayMessage()
{
   cout << "Hello from the function displayMessage.\n";
}
```

```
int main()
{
   cout << "Hello from main.\n"
   displayMessage();
   cout << "Back in function main again.\n";
   return 0;
}
```

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

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();`

Example program

```
1  // This program has three functions: main, First, and Second.
2  #include <iostream>
3  using namespace std;
4
5  // Function Prototypes
6  void first();
7  void second();
8
9  int main()
10 {
11    cout << "I am starting in function main.\n";
12    first();     // Call function first
13    second();    // Call function second
14    cout << "Back in function main again.\n";
15    return 0;
16 }
17
```

(Program Continues)

Example program cont…

```
18  //****************************************
19  // Definition of function first.        *
20  // This function displays a message.    *
21  //****************************************
22
23  void first()
24  {
25     cout << "I am now inside the function first.\n";
26  }
27
28  //****************************************
29  // Definition of function second.       *
30  // This function displays a message.    *
31  //****************************************
32
33  void second()
34  {
35     cout << "I am now inside the function second.\n";
36  }
```

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

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.

Example program

```
1  // This program demonstrates a function with a parameter.
2  #include <iostream>
3  using namespace std;
4
5  // Function Prototype
6  void displayValue(int);
7
8  int main()
9  {
10    cout << "I am passing 5 to displayValue.\n";
11    displayValue(5);  // Call displayValue with argument 5
12    cout << "Now I am back in main.\n";
13    return 0;
14 }
15
```

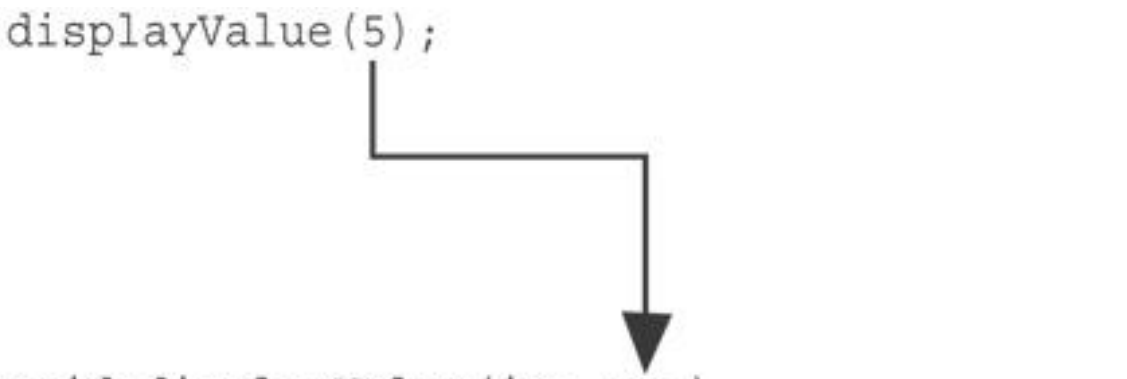
(Program Continues)

Example program cont…

```
16  //*********************************************************
17  // Definition of function displayValue.                   *
18  // It uses an integer parameter whose value is displayed. *
19  //*********************************************************
20
21  void displayValue(int num)
22  {
23     cout << "The value is " << num << endl;
24  }
```

Program Output

```
I am passing 5 to displayValue.
The value is 5
Now I am back in main.
```



The function call in line 11 passes the value 5 as an argument to the function.

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

Task

- Create a function countVowels(string s) that counts the number of vowels in a string.

Programming Fundamentals

Function in C++

- Sending data into a function
- Sending data into a function
- Passing multiple arguments
- Passing data by-value
- The return statement
- Lifetime of a variable

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.

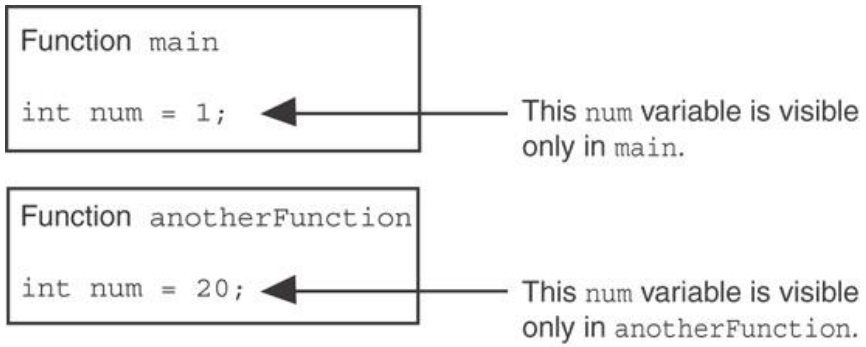
Example program (local variable life time or scope)

```
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
```

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.



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.
- You should avoid using global variables because they make programs difficult to debug.
- Any global that you create should be global constants.

Example program

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

```
1  // This program calculates gross p
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
```

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.

Use of the global variable in the program

- 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;
```

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.

Example program

```
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
```

```
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.

Using a Static Variable (a different Approach)

```
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
```

```
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 (a different Approach)

```
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
```

If you do initialize a local static variable, the initialization only happens once.

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.

Example program (scopes)

```
1  #include <iostream>
2  using namespace std;
3  void useLocal ( void ) ;       // function prototype
4  void useStaticLocal ( void ) ; // function prototype
5  void useGlobal ( void ) ;      // function prototype
6  int x = 1;     // global variable ( Declared outside of function; global variable with file scope.)
7  int main ()
8  {
9     int x = 5;   // local variable to main ( Local variable with function scope.)
10    cout << "local x in main's outer scope is " << x << endl;
11    { // start new scope
12       int x = 7; //Create a new block, giving x block scope. When the block ends, this x is destroyed.
13       cout << "local x in main's inner scope is " << x << endl;
14    }  // end new scope
15    cout << "local x in main's outer scope is " << x << endl;
16    useLocal () ;      // useLocal has local x
17    useStaticLocal () ; // useStaticLocal has static local x
18    useGlobal () ;     // useGlobal uses global x
19    useLocal () ;      // useLocal reinitializes its local x
20    useStaticLocal () ; // static local x retains its prior value
21    useGlobal () ;     // global x also retains its value
22    cout << "\nlocal x in main is " << x << endl;
23    return 0;  // indicates successful termination
24 }  // end main
```

```
local x in main's outer scope is 5
local x in main's inner scope is 7
local x in main's outer scope is 5
```

Example program (scope) cont…

```
25  // useLocal reinitializes local variable x during each call
26  void useLocal ( void)
27  {
28     int x = 25;  // initialized each time useLocal is called
29     // ( Automatic variable : local variable of function: This is destroyed when the function exits, and reinitialized when the function begins.)
30     cout << endl << "local x is " << x<< " on entering useLocal" << endl;
31     ++x;
32     cout << "local x is " << x << " on exiting useLocal" << endl;
33  }  // end function useLocal
34  // useStaticLocal initializes static local variable x only the
35  // first time the function is called; value of x is saved
36  // between calls to this function
37  void useStaticLocal ( void)
38  {// initialized only first time useStaticLocal is called
39     static int x = 50; //Static local variable of function; it is initialized only once, and retains its value between function calls.
40     cout << endl << "local static x is " << x << " on entering useStaticLocal" << endl;
41     ++x;
42     cout << "local static x is " << x << " on exiting useStaticLocal" << endl;
43  }  // end function useStaticLocal
44  // useGlobal modifies global variable x during each call
45  void useGlobal ( void)
46  {//This function does not declare any variables. It uses the global x declared in the beginning of the program.
47     cout << endl << "global x is " << x<< " on entering useGlobal" << endl;
48     x *= 10;
49     cout << "global x is " << x << " on exiting useGlobal" << endl;
50  }  // end function useGlobal
```

local x is 25 on entering useLocal
local x is 26 on exiting useLocal

local static x is 50 on entering useStaticLocal
local static x is 51 on exiting useStaticLocal

global x is 1 on entering useGlobal
global x is 10 on exiting useGlobal

Program output

Output by Calling each function first time

local x in main's outer scope is 5
local x in main's inner scope is 7
local x in main's outer scope is 5

local x is 25 on entering useLocal
local x is 26 on exiting useLocal

local static x is 50 on entering useStaticLocal
local static x is 51 on exiting useStaticLocal

global x is 1 on entering useGlobal
global x is 10 on exiting useGlobal

Output by Calling each function second time

local x is 25 on entering useLocal
local x is 26 on exiting useLocal

local static x is 51 on entering useStaticLocal
local static x is 52 on exiting useStaticLocal

global x is 10 on entering useGlobal
global x is 100 on exiting useGlobal

local x in main is 5

Programming Fundamentals

Function in C++

- Default arguments
- Passed by reference
- Functions Overloading
- Arrays in function

Default Arguments

- A Default argument is an argument that is passed automatically to a parameter if the argument is missing on the function call.
- Must be a constant declared in prototype:

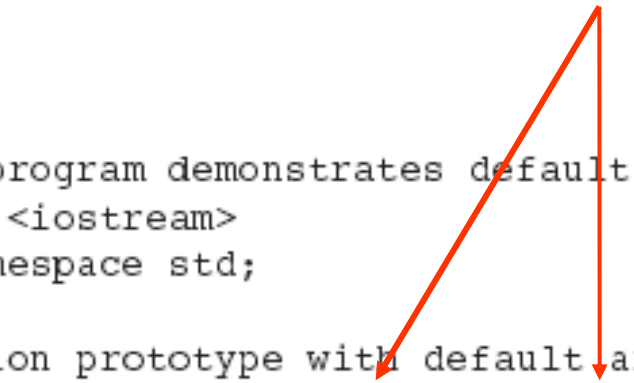
```
void evenOrOdd(int = 0);
```

- Can be declared in header if no prototype
- Multi-parameter functions may have default arguments for some or all of them:

```
int getSum(int, int=0, int=0);
```

Default Arguments (Example)

Default arguments specified in the prototype



```
1  // This program demonstrates default function arguments.
2  #include <iostream>
3  using namespace std;
4
5  // Function prototype with default arguments
6  void displayStars(int = 10, int = 1);
7
8  int main()
9  {
10    displayStars();      // Use default values for cols and rows.
11    cout << endl;
12    displayStars(5);     // Use default value for rows.
13    cout << endl;
14    displayStars(7, 3);  // Use 7 for cols and 3 for rows.
15    return 0;
16 }
```

(Program Continues)

Default Arguments (Example cont...)

```
18  //*********************************************************
19  // Definition of function displayStars.                   *
20  // The default argument for cols is 10 and for rows is 1.*
21  // This function displays a square made of asterisks.     *
22  //*********************************************************
23
24  void displayStars(int cols, int rows)
25  {
26     // Nested loop. The outer loop controls the rows
27     // and the inner loop controls the columns.
28     for (int down = 0; down < rows; down++)
29     {
30        for (int across = 0; across < cols; across++)
31           cout << "*";
32        cout << endl;
33     }
34  }
```

Program Output

```
**********

*****

*******
*******
*******
```

Example program

```
#include<iostream>
using namespace std;
int func(int x =10, int y=20, int z= 30);

int main () {
        cout<<"Passing no arguments during calling = "<<func()<<endl;
        cout<<"Passing one argument during calling = "<<func(50)<<endl;
        cout<<"Passing two arguments during calling = "<<func(50,50)<<endl;
        cout<<"Passing three arguments during calling = "<<func(50,50,50)<<endl;
}
int func(int a, int b, int c) {
        int sum = a + b+ c;
        return sum;
}
```

Output:

Passing no arguments during calling = 60
Passing one argument during calling = 100
Passing two arguments during calling = 130
Passing three arguments during calling = 150

Default Arguments (notes)

- If not all parameters to a function have default values, the defaultless ones are declared first in the parameter list:

```
int getSum(int, int=0, int=0);// OK
int getSum(int, int=0, int);  // NO
```

- When an argument is omitted from a function call, all arguments after it must also be omitted:

```
sum = getSum(num1, num2);     // OK
sum = getSum(num1, , num3);   // NO
```

Using Reference Variables as Parameters

- A mechanism that allows a function to work with the original argument from the function call, not a copy of the argument
- Allows the function to modify values stored in the calling environment
- Provides a way for the function to 'return' more than one value
- Passing by Reference
  - A reference variable is an alias for another variable
  - Defined with an ampersand (&)

```
void getDimensions(int&, int&);
```

  - Changes to a reference variable are made to the variable it refers to
  - Use reference variables to implement passing parameters *by* reference

Difference between pass by value and reference

Think of the coffee in the cup as the data in a variable.
One is a copy and one is the original

Example program

The & here in the prototype indicates that the parameter is a reference variable.

Here we are passing value by reference.

```
1  // This program uses a reference variable as a function
2  // parameter.
3  #include <iostream>
4  using namespace std;
5
6  // Function prototype. The parameter is a reference variable.
7  void doubleNum(int &);
8
9  int main()
10 {
11    int value = 4;
12
13    cout << "In main, value is " << value << endl;
14    cout << "Now calling doubleNum..." << endl;
15    doubleNum(value);
16    cout << "Now back in main. value is " << value << endl;
17    return 0;
18 }
19
```

(Program Continues)

Example program (cont…)

The & also appears here in the function header.

```
20  //*************************************************************
21  // Definition of doubleNum.                                   *
22  // The parameter refVar is a reference variable. The value    *
23  // in refVar is doubled.                                      *
24  //*************************************************************
25
26  void doubleNum (int &refVar)
27  {
28     refVar *= 2;
29  }
```

Program Output

```
In main, value is 4
Now calling doubleNum...
Now back in main. value is 8
```

Reference Variable Notes

- Each reference parameter must contain &
- Space between type and & is unimportant
- Must use & in both prototype and header
- Argument passed to reference parameter must be a variable – cannot be an expression or constant
- Use when appropriate – don't use when argument should not be changed by function, or if function needs to return only 1 value

Example program (return multiple values)

```
#include <iostream>
using namespace std;
 void compare(int a, int b, int &add_great, int &add_small)
{
   if (a > b) {
      add_great = a;
      add_small = b;
   }
   else {
      add_great = b;
      add_small = a;
   }
}
```

```
int main()
{
   int great, small, x, y;
 cout<<"Enter first number: ";
 cin>>x;
 cout<<"Enter second number: ";
 cin>>y;

   compare(x, y, great, small);
   cout<<"The greater number is "<< great << " and the
smaller number is "<<small;

   return 0;
}
```

Output:

```
Enter first number: 85
Enter second number: 95
The greater number is 95 and the smaller number is 85
```