CS1002 – Programming Fundamentals

Lecture # 24
Tuesday, November 22, 2022
FALL 2022
FAST – NUCES, Faisalabad Campus

Rizwan Ul Haq

Static Local Variables

- **Local variables**
  - Only exist while the function is executing
  - Are redefined each time function is called
  - Lose their contents when function terminates
- **static local variables**
  - Are defined with key word static

    static int counter;
  - Are defined and initialized **only the first time** the function is executed
  - Retain their contents between function calls
  - Better to initialize when declared

    static int counter = 0 ;

static variable illustrated

```
//Program: Static and automatic variables

#include <iostream>

using namespace std;

void test();

int main()
{
    int count;

    for (count = 1; count <= 5; count++)
        test();

    return 0;
}

void test()
{
    static int x = 0;
    int y = 10;

    x = x + 2;
    y = y + 1;

    cout << "Inside test x = " << x << " and y = "
         << y << endl;
}
```

Sample Run:

```
Inside test x = 2 and y = 11
Inside test x = 4 and y = 11
Inside test x = 6 and y = 11
Inside test x = 8 and y = 11
Inside test x = 10 and y = 11
```

Overloading Functions

- **Overloaded functions** are two or more functions that have the **same name**, but **different parameter lists**
- Can be used to create functions that perform the same task, but take **different parameter types** or **different number of parameters**
- **Compiler will determine** which version of function to call; by argument and parameter list
- Important thing is Formal parameter should be different either their datatype, number or position

Overloaded Functions - Examples

- If a program has these overloaded functions:

```
int getDimensions(int)              // 1
int getDimensions(int, int)         // 2
int getDimensions(int, double)      // 3
int getDimensions(double, double)   // 4
```

- The compiler will use them as follows:

```
int length, width;
double base, height;
getDimensions(length);              // 1
getDimensions(length, width);       // 2
getDimensions(length, height);      // 3
getDimensions(height, base);        // 4
```

Overloaded Functions - Examples

```
void functionXYZ()
void functionXYZ(int x, double y)
void functionXYZ(double one, int y)
void functionXYZ(int x, double y, char ch)
```

- Consider the following function headings to overload the function functionABC:

```
void functionABC(int x, double y)
int functionABC(int x, double y)
```

Default Arguments

- Values passed automatically if arguments are missing from the function call
- Must be a constant declared in prototype

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

- Multi-parameter functions may have default arguments for some or all of them

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

- If you specify a value to default parameter then it will be used otherwise default value will be used

Default Arguments

- If not all parameters to a function have default values, the ones without defaults must be declared first in the parameter list

```
int getSum(int, int=0, int=0);     // OK
int getSum(int, int=0, int);   // wrong!
```

- 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);  // wrong!
```

- **Constant** value **can't be assigned** to **reference parameter**

```
void func(int x, int& y=16, double z=34);
```

```
#include <iostream>
#include <iomanip>

using namespace std;

int volume(int l = 1, int w = 1, int h = 1);
void funcOne(int& x, double y = 12.34, char z = 'B');

int main()
{
    int a = 23;
    double b = 48.78;
    char ch = 'M';

    cout << fixed << showpoint;
    cout << setprecision(2);

    cout << "Line 1: a = " << a << ", b = "
         << b  << ", ch = " << ch << endl;          //Line 1
    cout << "Line 2: Volume = " << volume()
         << endl;                                   //Line 2
    cout << "Line 3: Volume = " << volume(5, 4)
         << endl;                                   //Line 3
    cout << "Line 4: Volume = " << volume(34)
         << endl;                                   //Line 4
    cout << "Line 5: Volume = "
         << volume(6, 4, 5) << endl;                //Line 5

    funcOne(a);                                     //Line 6
    funcOne(a, 42.68);                              //Line 7
    funcOne(a, 34.65, 'Q');                         //Line 8

    cout << "Line 9: a = " << a << ", b = "
         << b << ", ch = " << ch << endl;           //Line 9

    return 0;
}
```

```
int volume(int l, int w, int h)
{
    return l * w * h;                                   //Line 10
}

void funcOne(int& x, double y, char z)
{
    x = 2 * x;                                          //Line 11
    cout << "Line 12: x = " << x << ", y = "
         << y << ", z = " << z << endl;                 //Line 12
}
```

Sample Run:

```
Line 1: a = 23, b = 48.78, ch = M
Line 2: Volume = 1
Line 3: Volume = 20
Line 4: Volume = 34
Line 5: Volume = 120
Line 12: x = 46, y = 12.34, z = B
Line 12: x = 92, y = 42.68, z = B
Line 12: x = 184, y = 34.65, z = Q
Line 9: a = 184, b = 48.78, ch = M
```

The exit() Function

- Terminates execution of a program
- Can be called from any function
- Can pass a value to operating system to indicate status of program execution
- Usually used for abnormal termination of program
- Requires **cstdlib** header file

What will be the output of the following Program’s ?

```
void find(int a, int& b, int& c,)

int main()
{
    int one, two, three;

    one = 5;
    two = 10;
    three = 15;

    find(one, two, three);
    cout << one << ", " << two << ", " << three << endl;

    find(two, one, three);
    cout << one << ", " << two << ", " << three << endl;

    find(three, two, one);
    cout << one << ", " << two << ", " << three << endl;

    find(two, three, one);
    cout << one << ", " << two << ", " << three << endl;

    return 0;
 }

void find(int a, int& b, int& c)
{
    int temp;

    c = a + b;
    temp = a;
    a = b;
    b = 2 * temp;
}
```

```
5, 10, 15
20, 10, 15
25, 30, 15
45, 30, 60
```

```
int x;

void summer(int&, int);
void fall(int, int&);

int main()
{
    int intNum1 = 2;
    int intNum2 = 5;
    x = 6;

    summer(intNum1, intNum2);
    cout << intNum1 << " " << intNum2 << " " << x << endl;

    fall(intNum1, intNum2);
    cout << intNum1 << " " << intNum2 << " " << x << endl;
    return 0;
}

void summer(int& a, int b)
{
    int intNum1;
    intNum1 = b + 12;
    a = 2 * b + 5;
    b = intNum1 + 4;
}

void fall(int u, int& v)
{
    int intNum2;
    intNum2= x;
    v = intNum2 * 4;
    x = u - v;
}
```

```
15 5 6
15 24 -9
```

Array as a parameter to Functions

- Arrays are passed by reference only
- The symbol **&** is not used when declaring an array as a formal parameter
- The size of the array is usually omitted
- If provided, it is ignored by the compiler

E.g.

```
void foo(double firstList[], int secondList[])
{
    .
    .
    .
}
```

Arrays as Parameters to Functions

```
void initialize(int list[], int listSize)
{
   int count;
   for (count = 0; count < listSize; count++)
      list[count] = 0;
}
```

- The first parameter of the function initialize is an **int** array of any size
- When the function initialize is called, the size of the actual array is passed as the second parameter of the function initialize

Passing an Entire Array

- Use the array name, without any brackets, as the argument
- Can also pass the array size so the function knows how many elements to process

```
void printGrades(int[], int);      // prototype


void printGrades(int A[], int size) // header


printGrades(tests, 5);             // call
```

Constant Arrays as Formal Parameters

EXAMPLE 9-6

```
    //Function to initialize an int array to 0.
    //The array to be initialized and its size are passed
    //as parameters. The parameter listSize specifies the
    //number of elements to be initialized.
void initializeArray(int list[], int listSize)
{
    int index;

    for (index = 0; index < listSize; index++)
        list[index] = 0;
}
    //Function to print the elements of an int array.
    //The array to be printed and the number of elements
    //are passed as parameters. The parameter listSize
    //specifies the number of elements to be printed.
void printArray(const int list[], int listSize)
{
    int index;

    for (index = 0; index < listSize; index++)
        cout << list[index] << " ";
}
```

Base Address of an Array and Array in Computer Memory

- The base address of an array is the address, or memory location of the first array component
- If **list** is a one-dimensional array, its base address is the address of **list[0]**
- When we pass an array as a parameter, the base address of the actual array is passed to the formal parameter

Base Address of an Array and Array in Computer Memory (

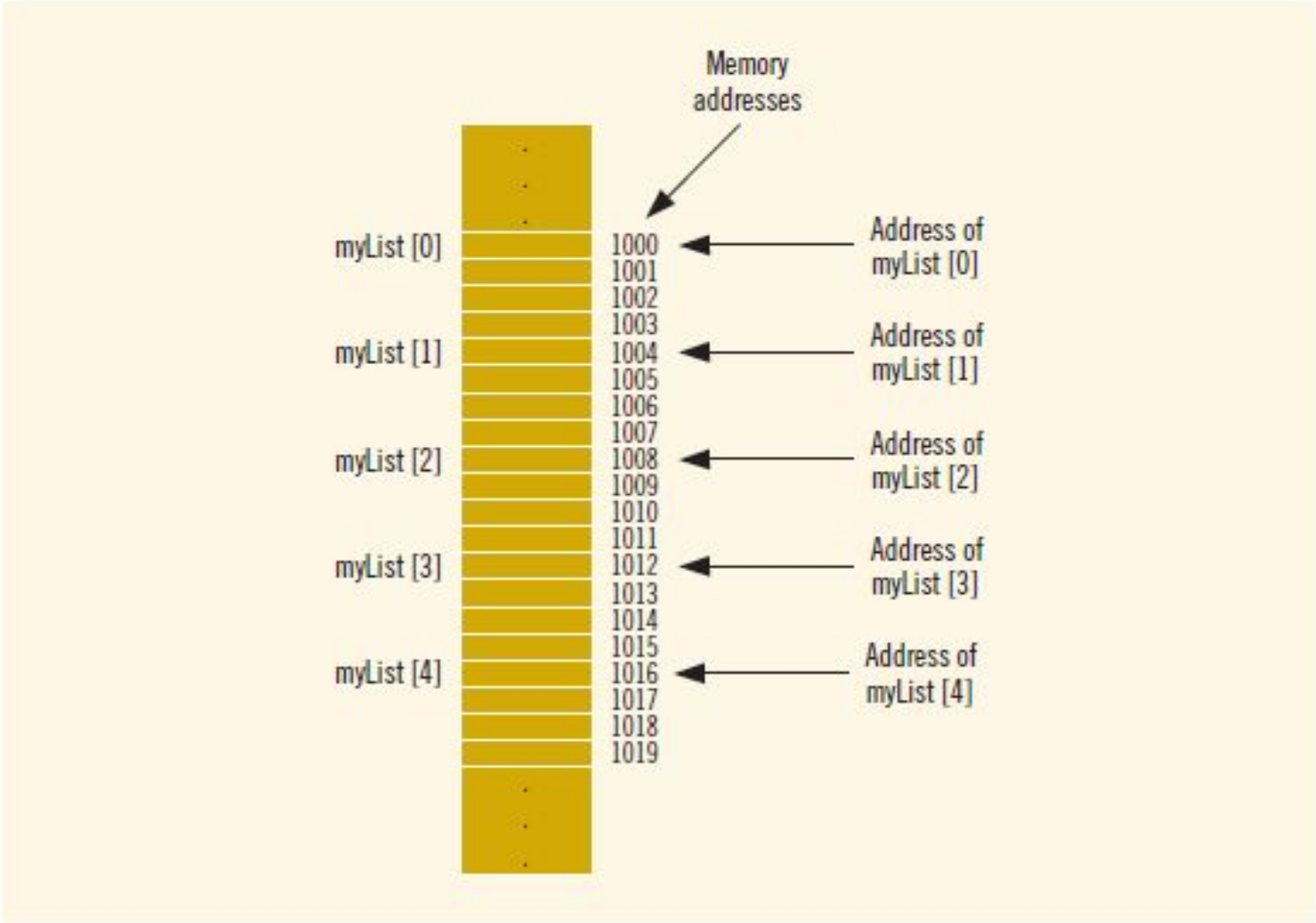

FIGURE 9-7 Array `myList` and the addresses of its components

Base Address of an Array and Array in Computer Memory (cont'd.)

- Consider the following statement:
  - cout << myList << endl; //Line 2
- This statement will not output the values of the components of **myList**
- In fact, the statement outputs the **value of myList**, which is the base address of the array
- This is why the statement will not generate a syntax error

Base Address of an Array and Array in Computer Memory (cont'd.)

- Suppose that you also have the following statement:

```
int yourList[5];
```

- Then, in the statement:

```
if (myList <= yourList)
```

- It does not determine whether the elements of **myList** are less than or equal to the corresponding elements of **yourList**

Base Address of an Array and Array in Computer Memory (cont'd.)

- when you declare an array, the only things about the array that the computer remembers are:
  - Name of the array
  - Its base address
  - The data type of each component
  - and (possibly) the number of components

Example

- Suppose you want to access the value of **myList[3]**
- Now, the base address of **myList** is 1000
  - Each component of **myList** is of type **int**, so it uses four bytes to store a value, and the index is 3
- To access the value of myList[3], the computer calculates the address

  $$1000 + 4 * 3 = 1000 + 12 = 1012$$

- That is, this is the starting address of myList[3]
  - So, starting at 1012, the computer accesses the next four bytes

Example

```
void arrayAsParameter(int list[], int size)
{
    ...
    list[2] = 28; //Line 4
    ...
}
```

- Suppose that you have the following call to this function:
- arrayAsParameter(myList, 5);
- list[2] = 28; This statement stores 28 into list[2]. To access list[2], the computer calculates the address as follows: 1000 + 4 * 2 = 1008

Functions Cannot Return a Value of the Type Array

- C++ does not allow functions to return a value of the type array

Searching an Array for a Specific Item

- Sequential search or linear search
  - Searching a list for a given item
  - Starting from the first array element
  - Compare **searchItem** with the elements in the array
  - Continue the search until either you find the item or no more data is left in the **list** to compare with **searchItem**

Searching an Array for a Specific Item (cont'd.)

```
int seqSearch(const int list[], int listLength, int searchItem)
{
    int loc;
    bool found = false;

    loc = 0;

    while (loc < listLength && !found)
        if (list[loc] == searchItem)
            found = true;
        else
              loc++;

    if (found)
        return loc;
    else
        return -1;

}
```

```
int seqSearch(const int list[], int listLength,
              int searchItem);                          //Line 4

int main()                                              //Line 5
{                                                       //Line 6
    int intList[ARRAY_SIZE];                            //Line 7
    int number;                                         //Line 8

    cout << "Line 9: Enter " << ARRAY_SIZE
         << " integers." << endl;                       //Line 9

    for (int index  = 0; index < ARRAY_SIZE; index++)   //Line 10
        cin >> intList[index];                          //Line 11

    cout << endl;                                       //Line 12

    cout << "Line 13: Enter the number to be "
         << "searched: ";                               //Line 13
    cin >> number;                                      //Line 14
    cout << endl;                                       //Line 15

    int pos = seqSearch(intList, ARRAY_SIZE, number);   //Line 16

    if (pos!= -1)                                       //Line 17
        cout <<"Line 18: " << number
             << " is found at position " << pos
             << endl;                                   //Line 18
    else                                                //Line 19
        cout << "Line 20: " << number
             << " is not in the list." << endl;         //Line 20

    return 0;                                           //Line 21
}                                                       //Line 22

//Place the definition of the function seqSearch
```

Questions