SINAI UNIVERSITY

CSW 232
Computer Programming (1)

SPRING 2024

Lecture 07 – User-Defined Functions II

Instructor: Dr. Tarek Abdul Hamid

Void Functions

- Void functions and value-returning functions have similar structures
  - Both have a heading part and a statement part
- User-defined void functions can be placed either before or after the function `main`
- If user-defined void functions are placed after the function `main`
  - The function prototype must be placed before the function `main`

Void Functions

- A void function does not have a return type
  - `return` statement without any value is typically used to exit the function early
- Formal parameters are optional
- A call to a void function is a stand-alone statement

Void Functions without Parameters

- Function definition syntax:

```
void functionName()
{
     statements
}
```

- `void` is a reserved word
- Function call syntax:

```
functionName();
```

Void Functions with Parameters

- Function definition syntax:

```
void functionName(formal parameter list)
{
    statements
}
```

- Formal parameter list syntax:

```
dataType& variable, dataType& variable, ...
```

- Function call syntax:

```
functionName(actual parameter list);
```

- Actual parameter list syntax:

```
expression or variable,expression or variable, ...
```

Void Functions with Parameters

```
void funexp(int a, double b, char c, int x)
{
    .
    .
    .
}
```

The function `funexp` has four parameters.

```
void expfun(int one, int& two, char three, double& four)
{
     .
     .
     .
}
```

The function `expfun` has four parameters: (1) `one`, a value parameter of type `int`; (2) `two`, a reference parameter of type `int`; (3) `three`, a value parameter of type `char`, and (4) `four`, a reference parameter of type `double`.

Void Functions with Parameters

- **Value parameter**: a formal parameter that receives a copy of the content of actual parameter

- **Reference parameter**: a formal parameter that receives the location (memory address) of the corresponding actual parameter

Value Parameters

- If a formal parameter is a value parameter
  - The value of the corresponding actual parameter is copied into it
- The value parameter has its own copy of the data
- During program execution
  - The value parameter manipulates the data stored in its own memory space

Reference Variables as Parameters

- If a formal parameter is a reference parameter
  - It receives the memory address of the corresponding actual parameter
- A reference parameter stores the address of the corresponding actual parameter
- During program execution to manipulate data
  - The address stored in the reference parameter directs it to the memory space of the corresponding actual parameter

Reference Variables as Parameters

- Reference parameters can:
  - Pass one or more values from a function
  - Change the value of the actual parameter
- Reference parameters are useful in three situations:
  - Returning more than one value
  - Changing the actual parameter
  - When passing the address would save memory space and time

Calculate Grade

```
//This program reads a course score and prints the
//associated course grade.

#include <iostream>
using namespace std;

void getScore(int& score);
void printGrade(int score);

int main()
{
    int courseScore;

    cout << "Line 1: Based on the course score, \n"
         << "    this program computes the "
         << "course grade." << endl;                    //Line 1

    getScore(courseScore);                              //Line 2

    printGrade(courseScore);                            //Line 3

    return 0;
}
```

Calculate Grade

```
void getScore(int& score)
{
    cout << "Line 4: Enter course score: ";          //Line 4
    cin >> score;                                    //Line 5
    cout << endl << "Line 6: Course score is "
         << score << endl;                           //Line 6
}

void printGrade(int cScore)
{
    cout << "Line 7: Your grade for the course is "; //Line 7

    if (cScore >= 90)                                //Line 8
        cout << "A." << endl;
    else if (cScore >= 80)
        cout << "B." << endl;
    else if(cScore >= 70)
        cout << "C." << endl;
    else if (cScore >= 60)
        cout << "D." << endl;
    else
        cout << "F." << endl;
}
```

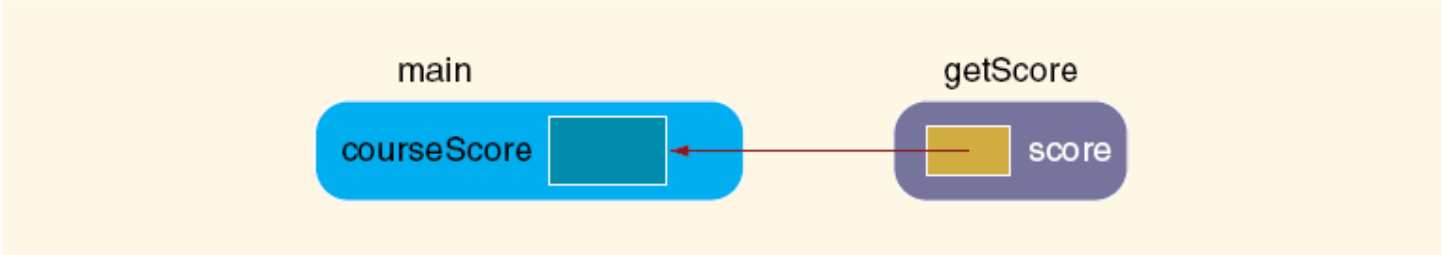


FIGURE 7-1 Variable courseScore and the parameter score

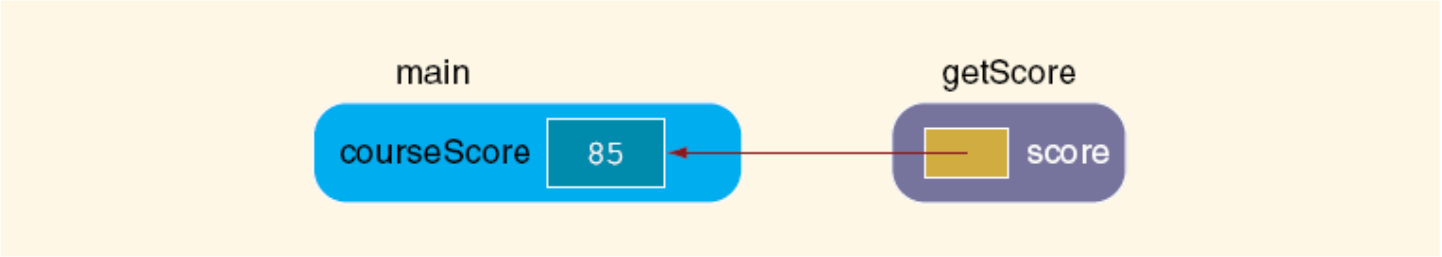


FIGURE 7-2 Variable courseScore and the parameter score after the statement in Line 5 executes

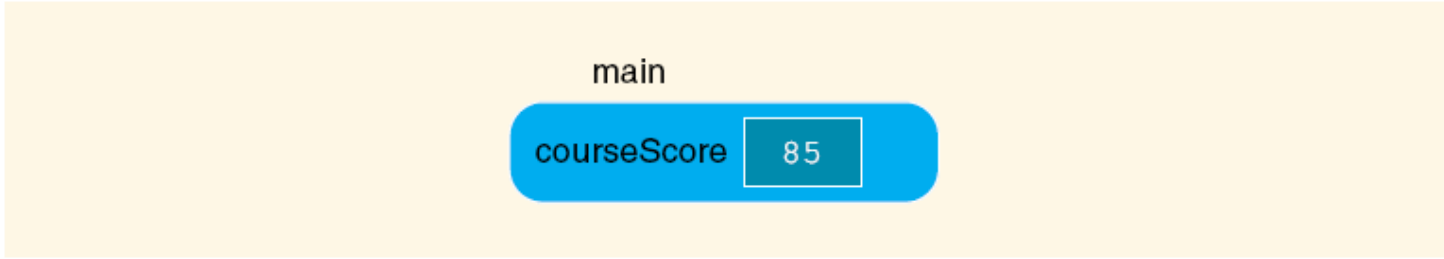
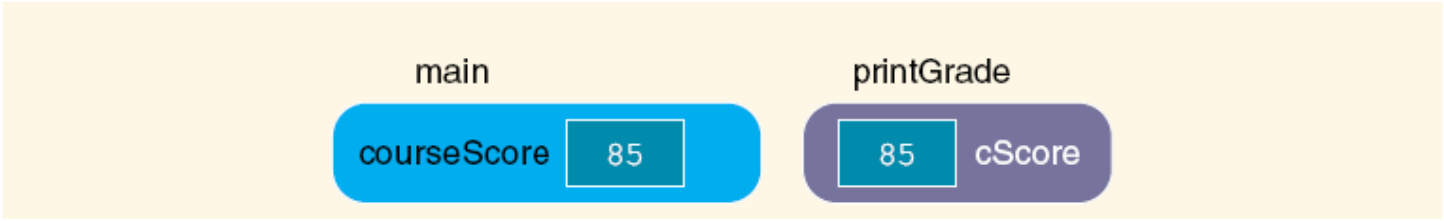


FIGURE 7-3 Variable courseScore after the statement in Line 6 is executed and control goes back to main



Value and Reference Parameters and Memory Allocation

- When a function is called
  - Memory for its formal parameters and variables declared in the body of the function (called local variables) is allocated in the function data area
- In the case of a value parameter
  - The value of the actual parameter is copied into the memory cell of its corresponding formal parameter

Value and Reference Parameters and Memory Allocation

- In the case of a reference parameter
  - The address of the actual parameter passes to the formal parameter
- Content of formal parameter is an address
- During execution, changes made by the formal parameter permanently change the value of the actual parameter
- Stream variables (e.g., `ifstream`) should be passed by reference to a function

```
#include <iostream>

using namespace std;

void funOne(int a, int& b, char v);
void funTwo(int& x, int y, char& w);

int main()
{
    int num1, num2;
    char ch;

    num1 = 10;                                          //Line 1
    num2 = 15;                                          //Line 2
    ch = 'A';                                           //Line 3

    cout << "Line 4: Inside main: num1 = " << num1
         << ", num2 = " << num2 << ", and ch = "
         << ch << endl;                                 //Line 4

    funOne(num1, num2, ch);                             //Line 5

    cout << "Line 6: After funOne: num1 = " << num1
         << ", num2 = " << num2 << ", and ch = "
         << ch << endl;                                 //Line 6

    funTwo(num2, 25, ch);                               //Line 7

    cout << "Line 8: After funTwo: num1 = " << num1
         << ", num2 = " << num2 << ", and ch = "
         << ch << endl;                                 //Line 8

    return 0;
}
```

```
void funOne(int a, int& b, char v)
{
    int one;

    one = a;                                              //Line 9
    a++;                                                  //Line 10
    b = b * 2;                                            //Line 11
    v = 'B';                                              //Line 12

    cout << "Line 13: Inside funOne: a = " << a
         << ", b = " << b << ", v = " << v
         << ", and one = " << one << endl;                //Line 13
}

void funTwo(int& x, int y, char& w)
{
    x++;                                                  //Line 14
    y = y * 2;                                            //Line 15
    w = 'G';                                              //Line 16

    cout << "Line 17: Inside funTwo: x = " << x
         << ", y = " << y << ", and w = " << w
         << endl;                                         //Line 17
}
```

Sample Run:

```
Line 4: Inside main: num1 = 10, num2 = 15, and ch = A
Line 13: Inside funOne: a = 11, b = 30, v = B, and one = 10
Line 6: After funOne: num1 = 10, num2 = 30, and ch = A
Line 17: Inside funTwo: x = 31, y = 50, and w = G
Line 8: After funTwo: num1 = 10, num2 = 31, and ch = G
```

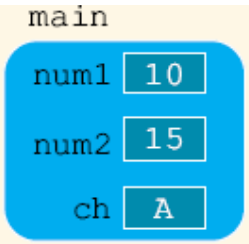


FIGURE 7-5 Values of the variables after the statement in Line 3 executes

```
one = a;                                    //Line 9
```

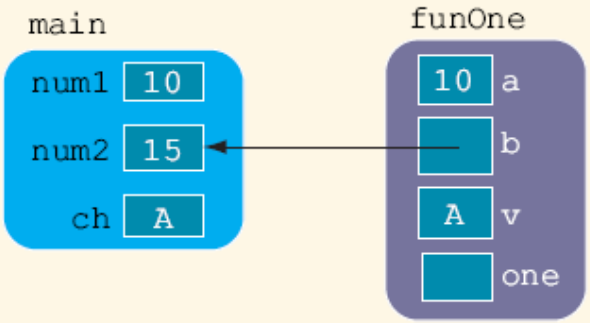
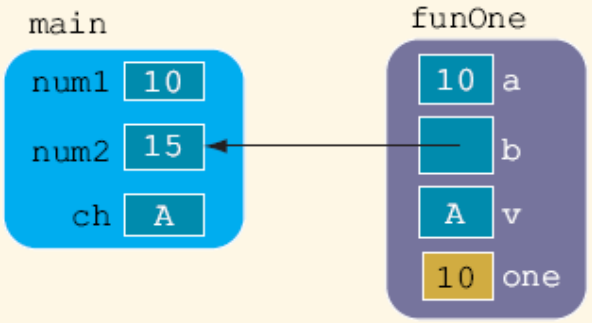


FIGURE 7-6 Values of the variables just before the statement in Line 9 executes



```
a++;                                    //Line 10
b = b * 2;                              //Line 11
v = 'B';                                //Line 12
```

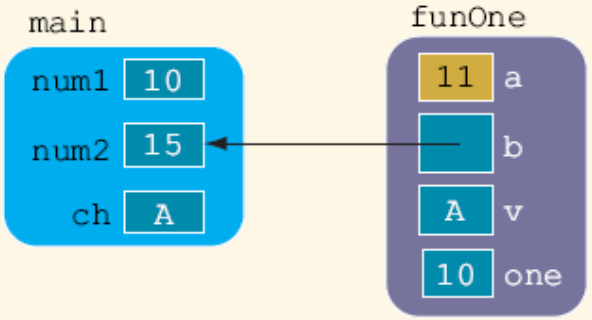


FIGURE 7-8 Values of the variables after the statement in Line 10 executes

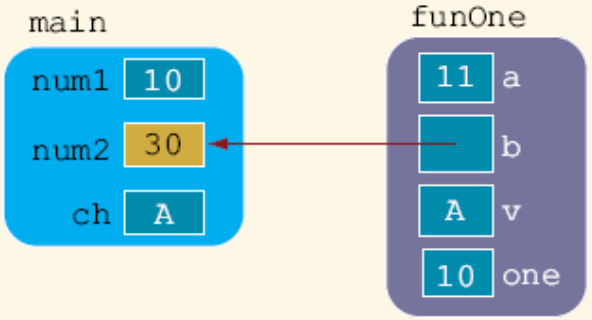
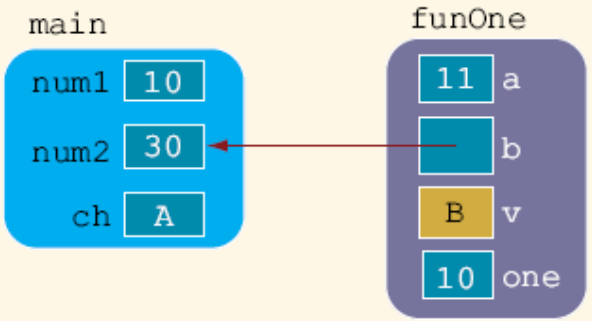


FIGURE 7-9 Values of the variables after the statement in Line 11 executes



```
cout << "Line 13: Inside funOne: a = " << a
     << ", b = " << b << ", v = " << v
     << ", and one = " << one << endl;          //Line 13
```

The statement in Line 13 produces the following output:

```
Line 13: Inside funOne: a = 11, b = 30, v = B, and one = 10
```

```
cout << "Line 6: After funOne: num1 = " << num1
     << ", num2 = " << num2 << ", and ch = "
     << ch << endl;                             //Line 6
```

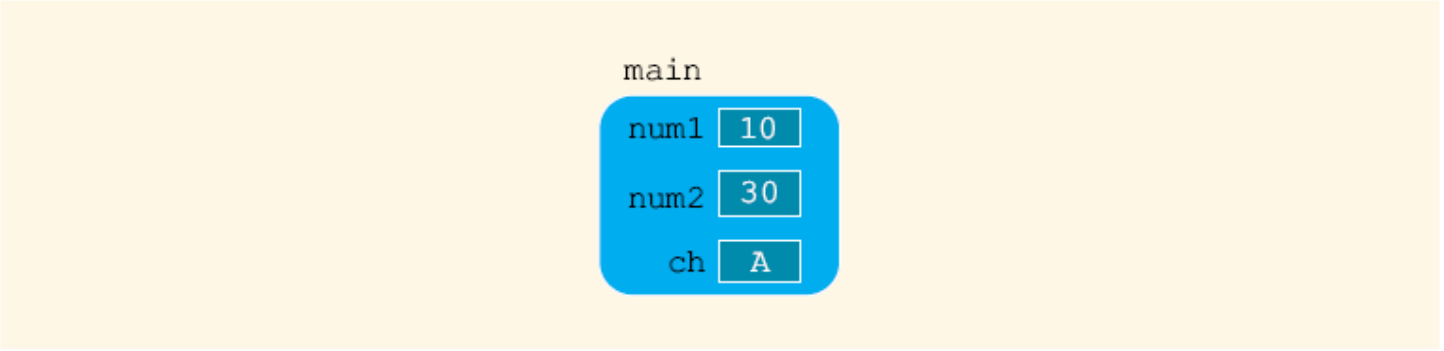


FIGURE 7-11 Values of the variables when control goes back to Line 6

Line 6 produces the following output:

```
Line 6: After funOne: num1 = 10, num2 = 30, and ch = A
```

```
x++;                                    //Line 14
y = y * 2;                              //Line 15
```

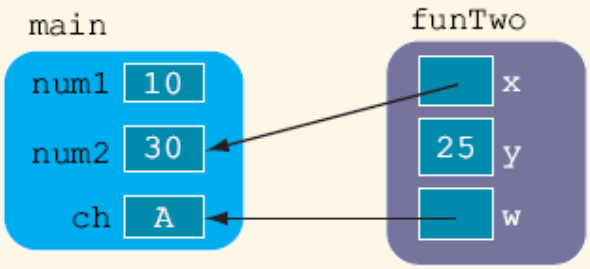


FIGURE 7-12 Values of the variables before the statement in Line 14 executes

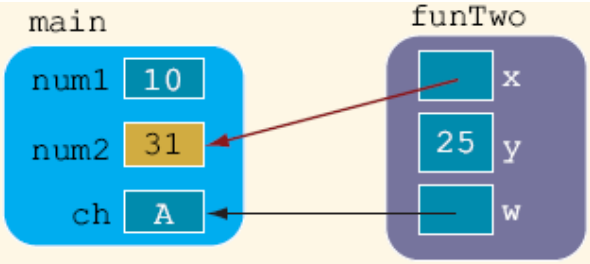
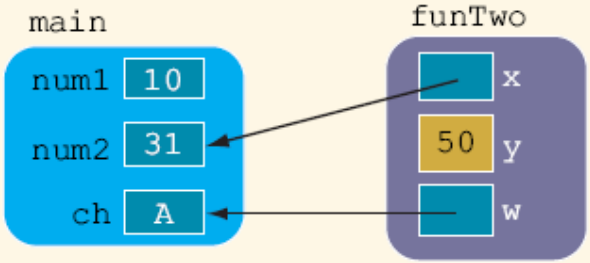


FIGURE 7-13 Values of the variables after the statement in Line 14 executes



```
w = 'G';                                                    //Line 16

cout << "Line 17: Inside funTwo: x = " << x
     << ", y = " << y << ", and w = " << w
     << endl;                                               //Line 17
```

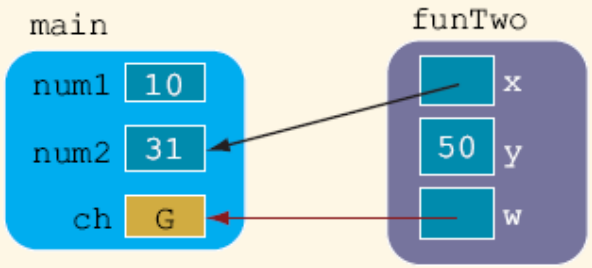
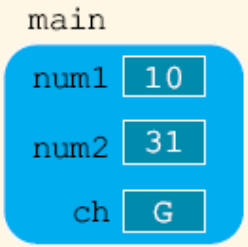


FIGURE 7-15 Values of the variables after the statement in Line 16 executes

Line 17 produces the following output:

```
Line 17: Inside funTwo: x = 31, y = 50, and w = G
```

```
cout << "Line 8: After funTwo: num1 = " << num1
     << ", num2 = " << num2 << ", and ch = "
     << ch << endl;                                         //Line 8
```




```
Line 8: After funTwo: num1 = 10, num2 = 31, and ch = G
```

Reference Parameters and Value-Returning Functions

- You can also use reference parameters in a value-returning function
  - Not recommended
- By definition, a value-returning function returns a single value
  - This value is returned via the return statement
- If a function needs to return more than one value, you should change it to a void function and use the appropriate reference parameters to return the values

Programming Example: Classify Numbers

- In this example, we use functions to rewrite the program that determines the number of odds and evens from a given list of integers
- Main algorithm remains the same:
  - Initialize variables, zeros, odds, evens to 0
  - Read a number
  - If number is even, increment the even count
    - If number is also zero, increment the zero count; else increment the odd count
  - Repeat Steps 2-3 for each number in the list

Programming Example: Classify Numbers

- The program functions include:
  - `initialize`: initialize the variables, such as zeros, odds, and evens
  - `getNumber`: get the number
  - `classifyNumber`: determine if number is odd or even (and whether it is also zero); this function also increments the appropriate count
  - `printResults`: print the results

```
void initialize(int& zeroCount, int& oddCount, int& evenCount)
{
    zeroCount = 0;
    oddCount = 0;
    evenCount = 0;
}
```

```
void getNumber(int& num)
{
    cin >> num;
}
```

```
void classifyNumber(int num, int& zeroCount, int& oddCount,
                    int& evenCount)
{
    switch (num % 2)
    {
    case 0:
        evenCount++;
        if (num == 0)
            zeroCount++;
        break;
    case 1:
    case -1:
        oddCount++;
    } //end switch
} //end classifyNumber
```

```
void printResults(int zeroCount, int oddCount, int evenCount)
{
    cout << "There are " << evenCount << " evens, "
         << "which includes " << zeroCount << " zeros"
         << endl;

    cout << "The number of odd numbers is: " << oddCount
         << endl;
} //end printResults
```

Programming Example: Main Algorithm

- Call `initialize` to initialize variables
- Prompt the user to enter 20 numbers
- For each number in the list
  - Call `getNumber` to read a number
  - Output the number
  - Call `classifyNumber` to classify the number and increment the appropriate count
- Call `printResults` to print the final results

```
int main()
{
        //Variable declaration
    int counter; //loop control variable
    int number;  //variable to store the new number
    int zeros;   //variable to store the number of zeros
    int odds;    //variable to store the number of odd integers
    int evens;   //variable to store the number of even integers

    initialize(zeros, odds, evens);                    //Step 1

    cout << "Please enter " << N << " integers."
         << endl;                                      //Step 2
    cout << "The numbers you entered are: "
         << endl;

    for (counter = 1; counter <= N; counter++)         //Step 3
    {
        getNumber(number);                             //Step 3a
        cout << number << " ";                         //Step 3b
        classifyNumber(number, zeros, odds, evens); //Step 3c
    } // end for loop

    cout << endl;

    printResults(zeros, odds, evens);                  //Step 4

    return 0;
}
```


Thanks!