Faculty of Engineering Technology

CET211 - Data Structures and Algorithms

Fall 2024 - Sheet #1

Instructors: Dr. Hesham Sakr

Eng. Aya Abdel Naby - Eng. Hager Sobeah

Topics Covered:

Introduction to C++

- Input/Output
- Variables and Data Types
- Control Structures (if conditions, loops)
- Arrays (Static and Dynamic)
- Functions

A. Introduction to C++ - Input/Output

In Python and Java, you use `print()` or `System.out.println()` for output and you use `input()` or `Scanner` for input. In C++, input and output are handled with:

- `std::cin` is used to receive input from the user.
- `std::cout` is used to display output on the console.

Both `cin` and `cout` are part of the iostream library which can be imported using the following line of code at the beginning of the program: `#include <iostream>`

Example Code (output):

```
#include <iostream>

int main() {

    std::cout << "Hello, World!" << std::endl;

    return 0;

}
```

*Note: cout outputs data to the console using the **insertion operator (<<)**, which can handle various types of data like strings, integers, and floating-point values.*

You can also use: `using namespace std;` as a shortcut to avoid having to prefix the standard library names (from the **std** namespace) with `std::` every time you use them.

Example Code (input & output):

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

int main() {
    int num;
    cout << "Enter a number: ";  // Output
    cin >> num;                  // Input
    cout << "You entered: " << num << endl;  // Output
    return 0;
}
```

*Note: cin reads input using the **extraction operator (>>)** and is usually followed by a variable to store the input.*

B. Introduction to C++ - Variables and Data types

In C++, Python and Java, variables are fundamental components that store data values. C++ is similar to Java where each variable has a specific data type that determines the kind of data it can hold, such as integers, floating-point numbers, characters, Boolean values or strings. Understanding data types is crucial because it affects the operations that can be performed on the data and the amount of memory used.

Key Concepts:

- Variables: Named storage locations in memory that hold data which can be manipulated throughout a program.
- Data Types: Defines the type of data a variable can hold. Common data types include:
  - **`int`**: For integer values.
  - **`float`** and **`double`**: For floating-point values.
  - **`char`**: For single characters.
  - **`bool`**: For Boolean values (`true` or `false`).
  - **`string`**: For sequences of characters.

Example Code:

```
#include <iostream>
#include <string>
using namespace std;
int main() {
    int age = 20;
    float height = 5.9;
    char grade = 'A';
    string name = "Ahmed";
    cout << "Name: " << name << ", Age: " << age << endl;
    cout << "Height: " << height << ", Grade: " << grade << endl;
    return 0;}
```

Exercise#1:

Create a program that calculates the area of a rectangle using width and height input by the user.

C. Introduction to C++ - Control Structures

Control structures in C++ are similar to those in Java and Python, though the syntax is more like Java. The key control structures include:

- **`if-else`** conditions.
- Loops: `for`, `while`, and `do-while`.

Example Code:

```
#include <iostream>
using namespace std;
int main() {
    int num;
    cout << "Enter a number: ";
    cin >> num;
    if (num > 0) {
        cout << "Positive" << endl;
    } else if (num < 0) {
        cout << "Negative" << endl;
    } else {
        cout << "Zero" << endl;
    }
    return 0;
}
```

Exercise#1:

Write a program that checks if a number is even or odd using `if-else`.

Exercise#2:

Implement a loop to print numbers from 1 to 10 using a `for` loop.

D. Arrays (Static and Dynamic)

Memory in C++ is divided into two main regions: the **stack** and the **heap**. Understanding these concepts is crucial when working with arrays, especially when comparing static and dynamic arrays.

Stack	Heap
- The stack is a region of memory used for static memory allocation. - Memory is managed automatically, meaning variables created on the stack are automatically freed when the function exits. - Stack memory is used for: • **Local variables** (like basic data types and static arrays). • **Function calls** (where function arguments and return addresses are stored). - The stack is limited in size, and if too much memory is allocated, a **stack overflow** can occur.	- The heap is used for **dynamic memory allocation**. Unlike the stack, the heap can grow in size as needed. - Variables in the heap are manually managed, meaning you must explicitly allocate and free memory (using new and delete in C++). - Dynamic arrays and objects that need to persist beyond the current function scope are typically stored in the heap. - Not freeing memory from the heap can lead to **memory leaks**.

Example Code:
int x = 10; // x is stored in the stack
	Example Code:
	int* ptr = new int(10); // dynamically allocated memory
	delete ptr; // free the memory when done

Arrays in C++ are similar to those in Java but have some differences in memory management due to how the stack and heap are used. There are two types of arrays:

Static Arrays	Dynamic Arrays
- Fixed in size - Allocated on the stack - The size of a static array must be known at compile time and cannot be changed after declaration.	- Allow size to change at runtime. - Typically managed using pointers and the `new` keyword to allocate memory on the heap.

Example#1 (Static Array)

```
#include <iostream>
using namespace std;
int main() {
    int numbers[5] = {1, 2, 3, 4, 5};
    for (int i = 0; i < 5; i++) {
        cout << numbers[i] << endl;
    }
    return 0;
}
```

Example#2 (Dynamic Array)

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

int main() {
    int n;
    cout << "Enter size of array: ";
    cin >> n;
    int* arr = new int[n];  // Dynamic allocation

    for (int i = 0; i < n; i++) {
        cout << "Enter value for element " << i+1 << ": ";
        cin >> arr[i];
    }

    cout << "You entered: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }

    delete[] arr;  // Free allocated memory
    return 0;
}
```

E. Functions

Functions in C++ are similar to Java methods or Python functions. They can return values and accept parameters.

Example Code:

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

int add(int a, int b) {
    return a + b;
}

int main() {
    int x = 5, y = 3;
    cout << "Sum: " << add(x, y) << endl;
    return 0;
}
```

Exercise#1:

Write a C++ function to count the number of occurrences of a given element in an array.

Exercise#2:

Write a C++ program to find the intersection of two arrays (common elements).

F. Extra Questions

1. Write a C++ function to reverse an array.
2. Write a C++ program to find the second largest element in an array.
3. Write a C++ program to merge two sorted arrays into a single sorted array, both have the same size.
4. Write a C++ function to rotate an array to the right by **k** positions.
5. Given a sorted array, write a C++ function to find an element.