CHAPTER

ONE

C++ LANGUAGE CONSTRUCTS: WEEK01

1.1 Simplistic C++ Programs

C++ allows procedural-style programs (just like C). Input and output is done using stream-operators

1.1.1 A Minimal C++ Program

Program `sample01.cpp` adds two integer numbers, prints their sum. Pay attention to the signature of `main()` method (it should returns integer); near the end there is an optional line `return 0` which means that the process exited normally.

```
#include <iostream>

using namespace std;
int main()
{
    cout << "Enter two numbers:" << endl;
    int v1 = 0, v2 = 0;
    cin >> v1 >> v2;
    cout << v1 << " + " << v2 << " = " << (v1 + v2) << endl;
    return 0;
}
```

Sample standard input (`STDIN`) for this program:

```
3 5
```

Sample standard output (`STDOUT`) for this program and the input:

```
Enter two numbers:
3 + 5 = 8
```

1.1.2 Reading Input with “while” Loop

Program `sample02.cpp` reads and adds all positive numbers on a single line; it ignores all input after the first negative number or zero (as well as all input on subsequent lines).

- It reads standard input line by line; command `getline` reads everything up to a newline symbol into a `string` variable `line`.
- It builds a string-stream object `lineStream` to read from this input.
- It starts reading integer numbers from this input (every integer ends with some whitespace).
- It exits the loop as soon as the end of the first line is reached (or there is a negative number in the input).

```
#include <iostream>
#include <sstream>
#include <string>

using namespace std;
int main()
{
    string line;
    getline(cin,line);
    stringstream lineStream(line);
    int num = 0, total = 0;
    while (lineStream >> num && num > 0) {
        total += num;
    }
    cout << "Total is " << total << endl;
    return 0;
}
```

Sample standard input (`STDIN`) for this program:

```
3 5 7 9 -1 3 3 3
2 4 6
```

Sample standard output (`STDOUT`) for this program and the input:

```
Total is 24
```

1.1.3 Reading Input with “for” Loop

Program `sample03.cpp` asks the user to input the count of numbers `n`. After that it reads all `n` integers and adds them up. Note that the `for` loop consists of three parts separated with semicolons (initialization of the loop variable, loop condition and increment that is done after each loop iteration).

```
#include <iostream>

using namespace std;
int main()
{
    int n = 0;
    cout << "Enter the count of numbers: " << endl;
    cin >> n;
    cout << "Enter " << n << " integers: " << endl;
    int total = 0;
```

(continues on next page)

(continued from previous page)

```
    for (int i = 0; i < n; i++) {
        int num;
        cin >> num;
        total += num;
    }
    cout << "The total of " << n << " numbers is " << total << endl;
    return 0;
}
```

Sample standard input (STDIN) for this program:

```
10
1 2 3 4 5 6 7 8 9 10
```

Sample standard output (STDOUT) for this program and the input:

```
Enter the count of numbers:
Enter 10 integers:
The total of 10 numbers is 55
```

1.2 Fixed-size Arrays

```
const int m = 3;
const int n = 4;
int arr[m][n]={{3, 8, 4, 6},
               {2, 7, 1, 5},
               {0, 9, -1, 2}};
```

1.3 Integer Data Types

Register overflow happens silently without warning:

```
int numbers[5] = {1000000001, 1000000002,
                  1000000003, 1000000004, 1000000005};
int sz = sizeof(numbers)/sizeof(int);  // assigns array size (=5).
int iTotal;                 // 4 byte register (signed)
long unsigned luTotal;      // 4 bytes register (unsigned)
long long llTotal;          // 8 bytes register (signed)
for (int i = 0; i < sz; i++) {
  intTotal += numbers[i];
      luTotal += numbers[i];
      llTotal += numbers[i];
      cout << i << "-th partial sum is " << intTotal << "(int), "
    << luTotal << "(long unsigned), "
    << llTotal << "(long long)." << endl;
}
```

1.4 Float Data Types

```
int success = 4;
int total = 7;
cout << "Wrong proportion: " << 1.0*(success/total) << endl;
cout << "Correct proportion: " << (1.0*success/total) << endl;
// output exactly 6 decimal places:
cout << fixed << setprecision(6) << (1.0*success/total) << endl;
```

1.5 Text Data Types

```
int n = 5;
void staircase(int n) {
  for (int i = 1; i <= n; i++) {
    string spaces(n-i,' ');
    string hashes(2*i, '#');
    cout << "'" << spaces << hashes << spaces << "'" << endl;
  }
}
```

This will output the following triangle:

```
'    ##    '
'   ####   '
'  ######  '
' ######## '
'##########'
```

1.6 Vectors

Vectors are list-like objects that can hold objects of the same type, for example, `vector<int>` means a vector of `int` variables (4-byte integer numbers). See Initialize a vector in C++.

```
// initialize a vector with the given elements
vector<int> primes{2,3,5,7,11,13,17,19,23,29};
```

```
int vectorSumAsArray(vector<int> v) {
  int result = 0;
  for (int i = 0; i < v.size(); i++) {
    result += v[i];
  }
  return result;
}
```

If you want to access vector without using "array syntax" (`v[i]`), you can use method `v.at(i)` to access i-th element of a vector. In this case we have a square-sized vector of vectors and add up the elements on its diagonal:

```
int diagonalDifference(vector<vector<int>> vv) {
  int diag = 0;
  for (int i = 0; i < arr.size(); i++) {
    diag += arr.at(i).at(i);
```

(continues on next page)

(continued from previous page)

```
  }
  return diag;
}
```

Here is another method to operate with vectors: create iterator and loop over it. For vectors it is not much different from the above method (accessing vector like an array). But for some data structures iterator is more flexible as it does not need to know the size of the data structure.

```
int vectorSumWithIter(vector<int> v) {
  int result = 0;
  for (vector<int>::iterator it = v.begin(); it != v.end(); ++it) {
    result += *it;
  }
  return result;
}
```

1.7 Exercises

1. Input file contains a positive array size n and after that there are n integers. Find the count of sign flips (when a positive number is immediately followed by a negative number or vice versa).

Input: `0 -2 0 -10 2 -1 0 0 3 2 -3`

Output: `3`

Explanation: `10` is followed by `2`, `2` is followed by `-1`, `2` is followed by `-3`.

2. Input file contains positive integers m and n; after that there are $m \cdot n$ integers - a rectangular matrix with m rows and n columns; the matrix is input row by row.

   Replace every matrix element a_{ij} by the smallest element of a submatrix $A'(i,j)$ (this submatrix has size $i \times j$; it is located in the upper left corner of the original matrix A.

Input:

```
3 4
3 8 4 6
2 7 1 5
0 9 -1 2
```

Output:

```
3 4
3 3 3 3
2 2 1 1
0 0 -1 -1
```

Is it possible to compute the result without allocating new memory to hold $O(mn)$ integers?

3. The input contains n, the count of numbers followed by n integers. Find the smallest integer and also find, how many numbers are equal to this smallest integer.