

Fundamental Computer Programming- C++ Lab(I)

LAB 3

Distribution of BMI Levels

Week 3, Fall 2022

International Bachelor Program in Informatics
College of Informatics
Yuan Ze University

Purposes

- Get familiar with the basic structure of a C++ program
- Get familiar with some control statements such as **while (..) {}**, **for (...)** {...}, etc.

A typical while loop

```
while (condition) {  
    ...  
    ...  
    ...  
}
```

← Loop body: the statements inside while loop

Concept of “iteration”: One iteration consists of statements executed from the beginning to the end of a loop. For example, Execution of 10 iterations mean that the body of loop is executed 10 times.

- Develop problem solving skills

while Statement

- Used typically when we **do not know how** many times a task should be repeatedly executed. Certainly, it can also be used when the number of times of a loop body executed is known.
 - **Sentinel-controlled** *while*
 - **Counter-controlled** *while*

Sentinel Controlled *while*

- A ***condition*** is evaluated every time to determine whether a *while* loop should be terminated. This condition serves as a sentinel (哨兵). If the value of the sentinel is evaluated to be true, the *while* loop continues and the loop body is executed. Otherwise, the while loop stops.

Example of Sentinel

```
// Determine class average of grades. When
// a grade of -1 is read, stop reading grade.
int main()
{
    int total; // sum of grades entered by user
    int gradeCounter; // number of grades
    entered

    int grade; // grade value
    double average;
    // decimal number for average

    // initialization phase
    total = 0; // initialize total
    gradeCounter = 0; // initialize loop counter

    // processing phase
    // prompt for input and read grade from user
    cout << "Enter grade or -1 to quit: ";
    cin >> grade; // input grade or sentinel value
```

```
// loop until sentinel value -1 read from user
while ( grade != -1 ) // while grade is not -1
{
    total = total + grade; // add grade to total
    gradeCounter = gradeCounter + 1;
    // increment counter

    // prompt for input and read next grade
    cout << "Enter grade or -1 to quit: ";
    // input grade or sentinel value
    cin >> grade;
} // end while

// termination phase
average = total / 10;
// display total and average of grades
cout << "\nTotal of all 10 grades is " <<
total << endl;
cout << "Class average is " << average <<
endl;
}
```

Counter-Controlled *while*

**// Determine class average of grades. There
// are 10 grades.**

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

**int main ()
{**

int total; // sum of grades entered by user
// number of the grade to be entered next
int gradeCounter;
int grade; // grade value entered by user
int average; // average of grades

**// initialization phase
total = 0;** // initialize total
**// initialize loop counter
gradeCounter = 1;**

// processing phase, looping 10 times

**while (gradeCounter <= 10) {
cout << "Enter grade: ";
cin >> grade; // input next grade
total = total + grade; // add grade to total
// increment counter by 1
gradeCounter = gradeCounter + 1;
} // end while**

// termination phase

**average = total / 10;
// display total and average of grades
cout << "\nTotal of all 10 grades is " << total <<
endl;
cout << "Class average is " << average << endl;
} // end main**

Which is better? Sentinel or counter control?

do {...} while (...)

// Will execute the loop body at least once

```
#include <iostream>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    int counter = 1; // initialize counter
```

```
    do
```

```
    {
```

```
        cout << counter << " "; // display counter
```

```
        counter++; // increment counter
```

```
    } while ( counter <= 10 ); // end do...while
```

```
        cout << endl; // output a newline
```

```
    } // end main
```

for Statement

- Best used when the number of times a task that repeats is known in advance. It is typically counter-controlled.

// Counter-controlled repetition with the **for** statement.

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

What is an iteration?

```
int main()
{
```

```
    int sum = 0;
```

```
    // for statement header includes initialization,
    // loop-continuation condition and increment.
```

Statement Execution order:

1st iteration: 1 2 3 4 5

2nd to last iteration: 1 2 3 4


```
        1           2 1           5 4
for ( int counter = 1; counter <= 10; counter++ ){
2 3 sum = sum + counter;
3 4 cout << counter << " " << sum << endl;
    } // end for
} // end main
```

Always the last statement being executed in each iteration.

Yet another Multiple Selection Statement:

Must be evaluated into a constant integer value. Now it is a character.

switch



```
switch ( grade ) {  
    case 'A': // grade was uppercase A  
    case 'a': // or lowercase a  
        aCount++; // increment aCount  
        break; // necessary to exit switch  
  
    case 'B': // grade was uppercase B  
    case 'b': // or lowercase b  
        bCount++; // increment bCount  
        break; // exit switch  
  
    case 'C': // grade was uppercase C  
    case 'c': // or lowercase c  
        cCount++; // increment cCount  
        break; // exit switch  
  
    case 'D': // grade was uppercase D  
    case 'd': // or lowercase d
```

```
        dCount++; // increment dCount  
        break; // exit switch  
  
    case 'F': // grade was uppercase F  
    case 'f': // or lowercase f  
        fCount++; // increment fCount  
        break; // exit switch  
  
    case '\n': // ignore newlines,  
    case '\t': // tabs,  
    case ' ': // and spaces in input  
        break; // exit switch  
  
    default: // catch all other characters  
        cout << "Incorrect letter grade entered."  
            << " Enter a new grade." << endl;  
        break; // optional; will exit switch anyway  
} // end switch
```

About writing if (...) else if (...) else

```
if (sGrade >= 80) {  
    cout << "A";  
    countA = countA + 1;  
}  
else if (sGrade >= 70) {  
    cout << "B";  
    countB = countB + 1;  
}  
else if (sGrade >= 60) {  
    cout << "C";  
    countC = countC + 1;  
}  
else {  
    cout << "F";  
    countF = countF + 1;  
}
```

```
if (sGrade >= 80) {  
    cout << "A";  
    countA = countA + 1;  
}  
else if (sGrade < 80 && sGrade >= 70) {  
    cout << "B";  
    countB = countB + 1;  
}  
else if (sGrade < 70 && sGrade >= 60) {  
    cout << "C";  
    countC = countC + 1;  
}  
else if (sGrade < 60) {  
    cout << "F";  
    countF = countF + 1;  
}
```

Which is better?

Integer Array

- **An integer array is a sequence of consecutive memory locations which can be used to store more than one integer value. For example,**

```
int bmiAry[10] = {0};
```

// declare an array of 10 locations for storing 10 integer values. It starts from location 0, then 1, 2, ..., 9. Every location is initialized with a value of zero.

```
for (int i=0; i<10; i++) {  
    bmiAry[i] = i;  
    cout << i << " " << bmiAry[i] << endl;  
}
```

What will be printed out after executing the above for loop?

Body Mass Index (BMI)

- ◆ Based on the materials of Lab 2, we can calculate a person's BMI (Body Mass Index) using the following formula.

$$BMI = Weight/Height^2$$

- ◆ A BMI value can be categorized into one of the following levels according to the BMI ranges specified in the following table.

BMI level	BMI range(kg/m ²)
Unreasonably small (USL)	BMI <10
Highly severely underweight (HSUW)	10 <= BMI <= 15
Severely underweight (SUW)	15 < BMI <= 16
Underweight (UW)	16 < BMI < 18.5
Normal (NW)	18.5 <= BMI <= 25
Overweight (OW)	25 < BMI <= 40
Obese (OB)	40 < BMI <= 50
Severely obese (SOB)	50 < BMI <= 60
Highly severely obese (HSOB)	60 < BMI <= 70
Unreasonably large (ULL)	BMI > 70

LAB 3: BMI Level Distribution

- Your program should accept weight and height values **repeatedly** until a weight or height value is smaller than or equal to 0.
- If a weight or height value is not valid, their BMI calculation should be canceled and the program should proceed to handle next data set. That is, the program should ignore an invalid data set. A data set consists of the weight and height of a person.
- If both weight and height are valid, calculate the BMI value and determine its BMI level.
- Increase the BMI level count for the BMI level calculated in the previous step.
- After processing all the data sets, find the levels that have the maximum count.

Input Format

- A data set is input from keyboard as shown below.

65.2 1.73

The first number is weight and the second number is height. They are both placed on the same line.

- Your program should continue reading data sets until a weight or height value smaller than or equal to 0 is read.
- Some input data sets are shown on the right. In each data set, the first member is weight and the second member is height. Note that the 11-th data set is not valid.

56 1.5
27 0.93
44 1.3
56 1.65
60 1.8
200 2.4
160 2.1
23 1.6
34 1.3
54 1.8
10 1.6
56 1.3
77 1.75
0 0

Constraints

- Weight should be greater than or equal to 20kg and less than or equal to 200kg.
- Height should be greater than or equal to 0.9m and less than or equal to 2.5m.
- If weight or height is not in the defined range, the data set is invalid and should be ignored.
- Had better use an integer array called **disBMI** to store the number of data sets that belong to the same BMI levels as follows:
 - `disBMI[0]`, `disBMI[1]`, ..., `disBMI[9]` are used to store the number of data sets respectively belonging to USL, HSUW, ..., ULL.
- At least one *while* loop, two *for* loops, and a *switch* statement must be used.

Output

• After processing all the input data sets,

- Print out the number of valid data sets for each BMI level.
 - The first line should be the BMI levels, starting from the level name USL, then the names of the levels with increasing BMI values. Proper separation between the names of two adjacent levels should be set.
 - The second line should be the numbers of the data sets whose BMI levels are classified as the levels being presented in the first line.
 - Place the right most digit of the number for a BMI level right just below the last character of the level name. For example, if there are 13 data sets belonging to SUW level, the digit 3 of 13 should be placed right just below the character W of SUW. For this, you may need use a function called `setw()`, referring to <https://en.cppreference.com/w/cpp/io/manip/setw> for details.
- Print out the total number of valid data sets (denoted by “**Head count**”), i.e., number of persons whose weight and height are valid, as shown below.
- Print out the levels that have the maximum number of data sets.

Example

Input:

56 1.5
27 0.93
44 1.3
56 1.65
60 1.8
200 2.4
160 2.1
23 1.6
34 1.3
54 1.8
10 1.6
56 1.3
77 1.75
0 0

Output

```
#   USL  HSUW  SUW  UW  NW  OW  OB  SOB  HSOB  ULL
#   1    0    0    1   4   6   0   0    0    0
# Maximum count: 6
# Level=  OW
# Head count = 12
```

Another Example

Given the following data sets:

65 1.79	65.4 1.66
56 1.9	39.6 1.45
89 1.6	66.6 1.76
48 1.8	140 1.6
65 1.9	120 1.77
56 2.7	110 1.9
88 1.7	135.8 1.87
96 1.85	67.8 1.68
38 1.65	63 1.7
84 1.6	55.5 1.58
78 2.15	

We should get the following output:

```
#   USL   HSUW    SUW    UW    NW    OW    OB    SOB   HSOB   ULL
#     0     2     1     2     7     7     0     1     0     0
# Maximum count: 7
# Level=  NW  OW
# Head count = 20
```

Follow All Requirements

- Input formats
- Output formats
- All constraints on input data, especially not accepting invalid inputs
- Coding styles
 - Avoiding using variables which do not have expressive power. That is, a variable name should carry the meaning of the matter in which the variable intends to represent.

If you don't follow the requirements, up to 50% of the points for your lab will be deduced.