Control Statements for C++ &

Inheritance (10th Lab)

1. Control Statements

- Pseudo code
  - Artificial, informal language used to develop algorithms
  - Used to “think out” a program before coding it
    - Easy to convert into a C++ program
  - Similar to everyday English
    - Only executable statements
    - No need to declare variables
  - Not executed on computers

- Only three control structures are needed
  - No goto statements
  - Three control structures
    - Sequence structure
      - Programs executed sequentially by default
    - Selection structures
      - if, if...else, switch
    - Repetition structures
      - while, do...while, for

<table>
<thead>
<tr>
<th>Keywords Keywords common to the C and C++ programming languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto break case char Const</td>
</tr>
<tr>
<td>Continue default do double Else</td>
</tr>
<tr>
<td>enum extern float for Goto</td>
</tr>
<tr>
<td>if int long register Return</td>
</tr>
<tr>
<td>short signed sizeof static Struct</td>
</tr>
<tr>
<td>switch typedef union unsigned Void</td>
</tr>
<tr>
<td>volatile while</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C++-only keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>and and_eq asm bitand Bitor</td>
</tr>
<tr>
<td>bool catch class compl const_cast</td>
</tr>
<tr>
<td>delete dynamic_cast explicit export False</td>
</tr>
<tr>
<td>friend inline mutable namespace New</td>
</tr>
<tr>
<td>not not_eq operator or or_eq</td>
</tr>
<tr>
<td>private protected public reinterpret_cast static_cast</td>
</tr>
<tr>
<td>template this throw true Try</td>
</tr>
<tr>
<td>typeid typename using virtual wchar_t</td>
</tr>
<tr>
<td>xor xor_eq</td>
</tr>
</tbody>
</table>
if...else double-selection statement

Dangling-else problem
- Compiler associates else with the immediately preceding if
- Example
  - if ( x > 5 )
    - if ( y > 5 )
      - cout << "x and y are > 5";
    - else 
      - cout << "x is <= 5";
- Compiler interprets as
  - if ( x > 5 )
    - if ( y > 5 )
      - cout << "x and y are > 5";
    - else
      - cout << "x is <= 5";
- Rewrite with braces ({})
  - if ( x > 5 )
    - { 
      - if ( y > 5 )
        - cout << "x and y are > 5";
      - }
    - else
      - cout << "x is <= 5";

repetition statement
-while: repeat loop until the condition is true
  
  while(true); // infinity loop
  while(i--); // if i==0, loop ended

-for: for(initialization; loopContinuationCondition; increment)
  
  for(int i = 0; i < 10; i++);

-do...while: After executing body once, check condition to decide whether repeat
it or not
```
  do{
    counter++;
  }while(counter<=10);
```

-break: exit the current loop
```
  do{
    counter++;
    if(counter == 5) break;
  }while(counter<=10);
```

-continue: go to the first of loop
```
  do{
    counter++;
    if(counter == 10) continue;
  }while(counter<=10);
```

- **Switch**
  - case: execute the case, when given value of switch statement is equal to corresponding case.
  - default: for undefined case, basically this part is executed.
  - break: exit statement of switch control block
- Example
  ```
  switch(grade){
    case 'A':
      aCount++;
      break;
    case 'B':
    case 'C':
      bCount++;
      break;
    default:
      cout << "Error" << endl;
  }
  ```

- **Assignment Operators**
  - Assignment expression abbreviations
Addition assignment operator
- Example
c = c + 3; abbreviates to c += 3;

Other assignment operators
- d -= 4 (d = d - 4)
- e *= 5 (e = e * 5)
- f /= 3 (f = f / 3)
- g %= 9 (g = g % 9)

Increment and Decrement Operators
- Increment operator ++
  - Increments variable by one
    - Example
      c++

- Decrement operator --
  - Decrements variable by one
    - Example
      c--

```cpp
#include <iostream>
using std::cout;
using std::endl;

int main()
{
    int c;
    c=5;
    cout<< c <<endl;
    cout<< c++ <<endl;
    cout<< c <<endl;
    cout<<endl;
    c=5;
    cout << c <<endl;
    cout << ++c <<endl;
    cout << c <<endl;
    return 0;
}
```
Logical Operator
-&&: Logical AND
   
   ```
   if(gender==1 && age>=65) seniorFemales++;
   ```

-||: Logical OR
   
   ```
   if((semesterAverage>=90) || (finalExam >= 90)) grade = 'A';
   ```

-!: Logical NOT
   
   ```
   if(!(grade == 'F')) grade++;;
   ```

<table>
<thead>
<tr>
<th>Operators</th>
<th>Associativity</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>()</td>
<td>left to right</td>
<td>parentheses</td>
</tr>
<tr>
<td>++ - -</td>
<td>left to right</td>
<td>unary (postfix)</td>
</tr>
<tr>
<td>static_cast&lt; type &gt;{}</td>
<td>left to right</td>
<td>unary (prefix)</td>
</tr>
<tr>
<td>&amp; &amp; !</td>
<td>right to left</td>
<td>multiplicative</td>
</tr>
<tr>
<td>* / %</td>
<td>left to right</td>
<td>additive</td>
</tr>
<tr>
<td>&lt;&lt; &gt;&gt;</td>
<td>left to right</td>
<td>insertion/extraction</td>
</tr>
<tr>
<td>&lt; &lt;= &gt; &gt;=</td>
<td>left to right</td>
<td>relational</td>
</tr>
<tr>
<td>== !=</td>
<td>left to right</td>
<td>equality</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>: ?</td>
<td>left to right</td>
<td>logical OR</td>
</tr>
<tr>
<td>= += -= *= /= %=</td>
<td>right to left</td>
<td>assignment</td>
</tr>
<tr>
<td>,</td>
<td>left to right</td>
<td>comma</td>
</tr>
</tbody>
</table>
2. Inheritance

- definition
  1) “Student class inherits Person class.”
  - Student class inherits all member functions and member variables in Person class
  - Student object has member in Student class and also has member in Person class
  - Person class = inheriting class = Super or Base class
  - Student class = inherited class = Sub or Derived class

- Object creation procedure of inheriting class
  1) Memory space allocation
  2) Execution of Base class constructor
  3) Execution of Derived class constructor

- Object deletion procedure of Inherited class
  - Execution of Derived class destructor
  - Execution of Base class destructor

Example 1

```cpp
class AAA{
    int a;
public:
    int GetA() { return a;}
    AAA( int _a=5) {a = _a;}
};

class BBB : public AAA{
    int b;
public:
    int GetB(){ return b;}
    BBB( int _b){b = _b;}
    void ShowData();
};

void BBB::ShowData(){
    cout<<'AAA : '<<GetA()<<endl;
}```
cout<<"BBB : "<<GetB()<<endl;
}

int main(void){
  BBB bbb(10);
  bbb.ShowData();
  system("pause>nul");
  return 0;
}

1) class BBB : public AAA
   ● BBB class inherits AAA class as public
2) cout<<"AAA : "<<GetA()<<endl;
   ● Undefined member function of AAA class can be used in BBB class.

- Member initializer

Example 2

class AAA{
public:
  AAA(){ cout<<"AAA() call"<<endl; }
  AAA(int i){ cout<<"AAA(int i) call"<<endl; }
};

class BBB : public AAA{
public:
  BBB(){ cout<<"BBB() call"<<endl; }
  BBB(int j){ cout<<"BBB(int j) call"<<endl; }
  // BBB(int j) : AAA(j) { cout<<"BBB(int j) call"<<endl; }  
};

int main(){
```cpp
cout<<"Object 1 Creation"<<endl;
BBB bbb1;

cout<<"Object 2 Creation"<<endl;
BBB bbb2(10);

system("pause>nul");
return 0;
}
```

Result 1. BBB(int j) case

➔ When BBB class constructor is called, AAA class constructor, that is inherits BBB class, should be executed preferentially.

➔ In this case, void constructor of AAA class is executed, because there is no declaration on which constructor to be called in Base class.

Result 2. BBB(int j) : AAA(j) case

➔ Member initialization declaration

➔ Call the constructor of AAA class that can receive j as a parameter.

➔ It can decide the constructor of Base class, which you want to call, through member initializer.
Derivation

<table>
<thead>
<tr>
<th>Kinds of Inheritance</th>
<th>Public Inheritance</th>
<th>Protected Inheritance</th>
<th>Private Inheritance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Class</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Public member</td>
<td>Public</td>
<td>Protected</td>
<td>Private</td>
</tr>
<tr>
<td>Protected member</td>
<td>Protected</td>
<td>Protected</td>
<td>Private</td>
</tr>
<tr>
<td>Private member</td>
<td>Can’t access</td>
<td>Can’t access</td>
<td>Can’t access</td>
</tr>
</tbody>
</table>

- Protected member
  - It can be shown as ‘private’ from outside, but ‘public’ in inheritance relationship.
  - Same as ‘private’ member, but access can be allowed only in inheritance relationship.

```cpp
class AAA{
private:
    int a;
protected:
    int b;
};

class BBB :public AAA{
public:
    void SetData(){
        a=10; // private member-> compile error
        b=20; // protected member-> OK
    }
};

int main(void){
    AAA aaa;
    aaa.a=10; // private member-> compile error
    aaa.b=20; // protected member-> compile error

    BBB bbb;
    bbb.SetData();
    return 0;
}
```
★ Practice 1
Error occurs in the following code. Guess what’s the reason.

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

class A{
public:
    void printA(){ cout << "!" << endl; }
};

class B : protected A{
public:
    void printB(){ cout << "?" << endl; }
};

void main(){
    A a;
    B b;
    a.printA();
    b.printA();
}
```

(Answer) b.printA(); //error.
Since this is protected inheritance, public member is protected, protected member is protected, private member can’t access. So b.printA() member function of class A is protected, it cannot access from main function.

★ Practice 2
The following class is Account class that is defined for the bank account information. MyAccount class that inherits Account class publicly should have a member variable that contains customer’s transit limitation information(trans_limit) besides other Account class member variables.

- Write the blank in MyAccount.cpp
- Define constructor function and ShowData function in MyAccount
Call the constructor of Base class through member initializer
- Define class by referencing main function and execution results

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

class Account
{
protected:
    char acc_num[50]; //Account number
    int balance; // balance

public:
    Account(char* num, int bal)
    {
        strcpy(acc_num, num);
        balance = bal;
    }
};

class MyAccount : public Account
{
    /* Blank */
};

int main(void)
{
    MyAccount acc("302-1234-5678-07", 5000, 10000);
    acc.ShowData();
    //getchar();
    return 0;
};
```

Result:
Practice 3

- Program that calculate the average from the score input
- Write a program which implements class of header file.
- Receive the score as a input until input score is -1.
- Average value can show the two decimal places.
- In case of no input value, print the error message.

(Note)

- Configuration of print format and decimal places
  ```cpp
  #include <iomanip>
  using std::setprecision;
  ```

- `setprecision(int n)` : decide ‘n’ as the decimal number

- `static_cast<double>(total)`
  - C++ programming style. same as `(double)total` in C

- `cout << "Class average is " << setprecision(2) << fixed << average << endl;`
  - print the two decimal places

```cpp
//GradeBook.h

#include <string>
using std::string;

//GradeBook class definition
class GradeBook
{

public:

    GradeBook(string);

    void setCourseName(string);

    string getCourseName();

};
```
void displayMessage();
void determineClassAverage();

private:
    string courseName;
};
// end class Gradebook

//GradeBook.cpp
#include <iostream>
using std::cout;
using std::cin;
using std::endl;
using std::fixed;
#include <iomanip>
using std::setprecision;
#include "GradeBook.h"
/* Blank */

//main.cpp
#include "GradeBook.h"

//start main
int main()
{
    GradeBook myGradeBook( "Computer Programming" );

    myGradeBook.displayMessage();
    myGradeBook.determineClassAverage();
    return 0;
}
//end main

Result:
Welcome to the gradebook for Computer Programming
Enter grade or -1 to quit: 97
Enter grade or -1 to quit: 88
Enter grade or -1 to quit: 72
Enter grade or -1 to quit: -1

Total of all grades is 257
Class average is 85.67