INM359 Object-Oriented Programming in C++
2025-26 Coursework-1

To be done individually

Don't "help" others – they can help themselves by studying the relevant material

Contents

Setting Up

First, make sure you have a ***proper*** (command line-based) environment to develop and test your code. Use whichever IDE you want but your code ***MUST*** work with g++ version >=14 and the Makefile described in the following instructions.

Detailed instructions for installing a proper command-line environment are available at:
https://www.staff.city.ac.uk/c.kloukinas/cpp/00-setup/cpp-command-line-environment-setup.html

You need to use and submit the Makefile from the above instructions!

Standalone Questions

"Session 2 – Classes in C++ & Session 4 – Genericity, Containers"

Q1. Consider the following code. Fix the yellow parts, without modifying the parts in grey, so that the code compiles and runs correctly.

[3]

```
#include <iostream>
using namespace std;     // File: qQ1.cc
class person {
  int age;
 public:
  person(int a = 10) : age(a) {}
  int  get_age() const;
  void set_age(int a);
};
/* Your code here */
                  //
                  //
                  //
int main() {
  person p1;
  p1.set_age(25);
  cout << p1.get_age() << endl; // should print 25
  return 0;
```

```
}
```

Q2. Consider this generic person class variant. Fix the yellow parts, without modifying the parts in grey, so that the code compiles and runs correctly.

[4]

```
#include <iostream>
using namespace std;     // File: qQ2.cc
template <typename T>
class person {
  T age;
 public:
                  //
  person(T a) : age(a) {}
  T  get_age() const;
  void set_age(T a);
};
/* Your code here */
                  //
                  //
                  //
int main() {
  person<unsigned long> p1;
  p1.set_age(25);
  cout << p1.get_age() << endl; // should print 25
  return 0;
}
```

Q3. Now consider this final person class variant. Complete the main so that it prints 25 after having printed 10, without adding more methods to class person or changing the visibility of its members.

[3]

```
#include <iostream>
using namespace std;     // File: qQ3.cc
class person {
  int age;
 public:
  person(int a = 10) : age(a) {}
  int  get_age() const;
};
/* Your code here */
int main() {
  person p1;
  cout << p1.get_age() << endl; // should print 10
/* Your code here */
  cout << p1.get_age() << endl; // should print 25
  return 0;
}
```

Q4. Consider the following use of class my_array_class. Fix the code in the class so that the greyed-out code compiles and runs correctly.
Note: you should not modify any of the code in grey.

[1]

```
#include <iostream>
using namespace std;     // File: qQ4.cc
```

```
class my_array_class {
  size_t len = 3;
  int *a = new int [3];
 public:
  my_array_class() { a[0] = 1; a[1] = 2; a[2] = 3; }
  my_array_class(size_t ln, const int *o) : len(ln), a(new int [ln])
    { for (size_t n=0; n<len; ++n) a[n] = o[n]; }
 ~my_array_class() { delete[] a; }
 size_t get_length() { return len; }
 int get(size_t n) {return a[n];}
 int set(size_t n, int v) { int tmp = a[n]; a[n] = v; return tmp; }
};

void foo( const my_array_class & a2, size_t i ) {
   if (i < a2.get_length())
     std::cout <<  a2.get(i) << std::endl;
}

int main() {
  int zero12[] = {13, 1, 2};
  my_array_class a1(3, zero12);
  foo(a1, 0); // should print 13
  return 0;
}
```

"Session 3 – Overloading"

Q5. Extend the code in the class my_array_over below so that the greyed-out code compiles and runs correctly. (**Hint:** compare the greyed-out code here with Q4's.)

[2]

```
#include <iostream>
using namespace std;       // File: qQ5.cc
class my_array_over {
  size_t len = 1;
  int *a = new int [1];
 public:
  my_array_over() { a[0] = 0; }
  my_array_over(size_t ln, const int *o) : len(ln), a(new int [ln])
    { for (size_t n=0; n<ln; ++n) a[n] = o[n]; }
  ~my_array_over() { delete[] a; }
  /* Put your code here */
};
void foo( const my_array_over & a2, size_t i ) {
   if (i < a2.get_length())
     std::cout << a2[i] << std::endl;
}
int main() {
  int zero12[] = {23, 1, 2};
  my_array_over a1(3, zero12);
  a1.set(0, 13);
  foo(a1,0); // should print 13
 return 0;
}
```

Q6. Now do the same as in Q5 for the following code.

[2]

```
#include <iostream>
using namespace std;         // File: qQ6.cc
class my_array_over2 {
  size_t len = 1;
```

```
  int *a = new int [1];
 public:
  my_array_over2() { a[0] = 0; }
  my_array_over2(size_t ln, const int *o) : len(ln), a(new int [ln])
    { for (size_t n=0; n<ln; ++n) a[n] = o[n]; }
  ~my_array_over2() { delete[] a; }
/* Put your code here */
};

void foo( const my_array_over2 & a2, size_t i ) {
   if (i < a2.get_length())
     std::cout << a2[i] << std::endl;
}

int main() {
  int zero12[] = {13, 1, 2};
  my_array_over2 a1(3, zero12);
  a1[1] = 14; /* EXTRA LINE!!! */
  foo(a1,1); // should print 14
 return 0;
}
```

Q7. Consider the class m2dvector below, used for holding 2-dimensional (mathematical) vectors.

Q7.1. Overload the output operator, so that you can print objects of class m2dvector.

[6]

Q7.2. Overload the input operator, so that you can read objects of class m2dvector.

[12]

```
#include <iostream>
#include <string>
#include <sstream>
#include <vector>
#include <exception>
using namespace std;          // File: qQ7.cc
class m2dvector {
  vector<int> vi = {0, 0};
 public:
  m2dvector(vector<int> some_vi) : vi(some_vi) {if (some_vi.size() > 2)
throw std::invalid_argument("Input vector should have size 2."); }
  const vector<int> & get_data() const { return vi; }
};
/* Put your code here */

int main() {
  try {
    vector<int> vi = {11, 12, 13, 14, 15};
    m2dvector z1({vi[0], vi[1]});
    m2dvector z2({vi[3], vi[4]});
    cout << z1 << endl; /* should print: <11 12> */
    cout << z2 << endl; /* should print: <14 15> */
    string s1 = "<11 12>    <13 14>a"; /* s1 is a *single* test - code
                                          must work in general */
    istringstream iss(s1);
    iss >> z2 >> z1;
    cout << z1 << endl; /* should print: <13 14> */
```

```
    cout << z2 << endl; /* should print: <11 12> */
    char c;
    iss >> c; cout << c << endl; /* should print 'a' - the char after
the 2nd m2dvector above. */
    ostringstream oss;
    oss << z2;
    string so = oss.str();
    for ( char c : so )
      if (c == '\n') {cout << "Output contains a newline\n"; return 0;}
    cout << "Output does not contain a newline\n";
  } catch (...) { throw; }
  return 0;
}
```

Attention to detail – Submitting

You need to submit ***code*** *(source code, not executables!)*, *not* a report in Word/PDF/etc.
So, create a folder named after your student ID (0123456789) and **copy** in there the files q1.cc, q2.cc, …, along with the Makefile from page 1 (described in the environment setup instructions web page).
Zip your folder *(use zip, not rar/tar/7zip/etc.!)* – on the terminal that's done with:
`zip -r 0123456789.zip 0123456789/`
Unzip your zip file **somewhere else** (e.g., /tmp):
`cp 0123456789.zip /tmp ; cd /tmp ; unzip 0123456789.zip ; cd 0123456789`
Try to compile all your files there – do they compile with the provided Makefile (hopefully with no errors/warnings)?
If so, submit your zip file (which should contain the Makefile and the source files only – no binary code!).

10%

A viva maybe conducted to assess whether the coursework was completed sincerely by the student, without undue reliance on AI assistance.

Note: if you fail to submit on Moodle on time (or submit a wrong/corrupted zip file), then we *may* accept files submitted on a private repository on gitlab/github ***before*** the deadline. So, keep your code in a private git repository on gitlab/github.

Marking

There are **33** marks overall. These are scaled to contribute **30 marks** towards the **Coursework 1** total as follows:

- **27 marks** (i.e., 90% of 30) come from your **scaled performance on the programming tasks**.
- **3 marks** (i.e., 10%) are reserved for **"attention to detail"** — covering viva, formatting, code clarity, naming, and general polish.
- The **first 10 raw marks** are **fully credited** (1:1). Any **marks beyond 10** are **scaled down linearly**, so that the total **33 raw marks** map to a maximum of **27 scaled marks** (used in the coursework mark).
- The final Coursework 1 mark = **Scaled Programming Marks (out of 27)** + **Attention to Detail (up to 3)** = **Out of 30**

The first **10** marks earned count as a percentage, while the rest are weighted – see below.

```
cat > marks2025.cc << EOF
#include <iostream>     // std::cout
#include <algorithm>    // std::min
using namespace std;
int main() {
 float marks;
 const float threshold = 10.0;
 const float mx = 33.0;
 const float weight = (27 - threshold) / (mx - threshold);
 while (cin >> marks)
   cout // using the ite (?:) operator (ite = if-then-else)
    << ((marks<=threshold) ? marks : (threshold+(marks-
threshold)*weight))
    << char(9) // or, using min & max:
    << max( min(marks, threshold), min(marks, threshold) + (marks-
threshold)*weight )
    << endl;
 return 0;
}
EOF
make marks2025
./marks2025
```