Sample Exam Week 05

CSE 232 (Introduction to Programming II)

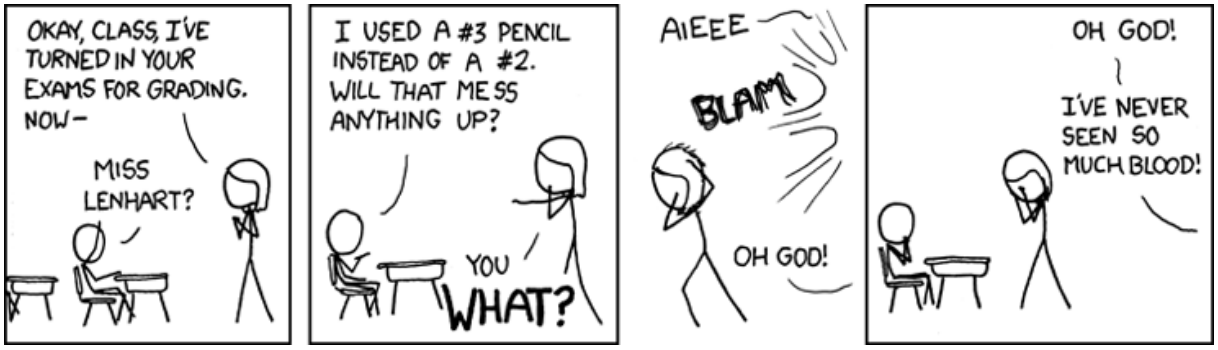
VERSION A

Full Name: .....

Student Number: .....

Instructions:

- DO NOT START/OPEN THE EXAM UNTIL TOLD TO DO SO.
- You may however write and bubble in your name, student number and exam **VERSION/FORM NUMBER** (with a #2 pencil) on the front of the printed exam and bubble sheet prior to the exam start. This exam is Version A. Your section doesn't matter and can be ignored.
- Present your MSU ID (or other photo ID) when returning your bubble sheet and printed exam.
- Only choose one option for each question. Please mark the chosen option in both this printed exam and the bubble sheet.
- Assume any needed `#include`s and `using std::...;` namespace declarations are performed for the code samples.
- Every question is worth the same amount of points. There are 55 questions, but you only need 50 questions correct for a perfect score.
- No electronics are allowed to be used or worn during the exam. This means smart-watches, phones and headphones need to be placed away in your bag.
- The exam is open note, meaning that any paper material (notes, slides, prior exams, assignments, books, etc.) are all allowed. Please place all such material on your desk prior to the start of the exam, (so you won't need to rummage in your bag during the exam).
- If you have any questions during the exam or finish the exam early, please raise your hand and a proctor will attend you.



http://xkcd.com/499/

1. What will be the output of the following code?

```
void foo(int *p) {
  cout << *p << endl;
  (*p)++;
}
int main() {
  int i = 10;
  foo(&i);
}
```

   (a) 10
   (b) Some garbage value
   (c) 11
   (d) Compile time error
   (e) Run time error
   (f) Segmentation fault/code crash

2. What will be the output of the following code?

```
void foo(int *p) {
  cout << *p << endl;
}
int main() {
  int i = 10, *p = &i;
  foo(p++);
}
```

   (a) Compile time error
   (b) 10
   (c) Some garbage value
   (d) Segmentation fault/code crash
   (e) Run time error
   (f) 11

3. What will be the output of the following code?

```
void foo(int *p) {
  cout << *p << endl;
}
int main() {
  int i = 10, *p = &i;
  foo(++p);
}
```

   (a) Some garbage value or error
   (b) 11
   (c) Compile time error
   (d) 10

4. What will be the output of the following code?

```
void foo(int *p) {
  p = new int(2);
  cout << *p << ' ';
}
int main() {
  int i = 97, *p = &i;
  foo(&i);
  cout << *p << endl;
}
```

   (a) Compile time error
   (b) Segmentation fault/code crash
   (c) Run time error
   (d) 2 97
   (e) 2 2

5. What will be the output of the following code?

```
void foo(int **p) {
  *p = new int(2);
  cout << **p << ' ';
}
int main() {
  int i = 97, *p = &i;
  foo(&p);
  cout << *p << endl;
}
```

   (a) Segmentation fault/code crash
   (b) 2 97
   (c) Run time error
   (d) 2 2
   (e) Compile time error

6. What will be the output of the following code?

```
int main() {
   int a[3] = {1, 2, 3};
   int *p = a;
   cout << p << '-' << a;
}
```

   (a) Two different addresses are printed
   (b) Same address is printed twice
   (c) Compile time error
   (d) Run time error
   (e) undefined

7. What is the purpose of a breakpoint in a debugger?

   (a) It causes the `while`, `for`, or `switch` statement to cease its execution and continue on the next line.
   (b) It pauses the execution of the program when it is encountered.
   (c) It intentionally crashes the program so that a core dump of useful information is produced.
   (d) It indicates a place where the program is performing undefined behavior or causing a compiler time error.

8. When running gdb, what does the "up" command do? This was covered in lab.

   (a) It adds the current value to the stack of accessed variables.
   (b) It moves you to the previously executed statement.
   (c) It prints the backtrace.
   (d) It brings you to the function call that invoked the current function.

9. When compiling code for running with `gdb`, what command line argument should be used?

   (a) -o
   (b) -c
   (c) -x
   (d) -d
   (e) -g

10. What is the primary downside to using `#pragma once` instead of the traditional header guards?

    (a) It interferes with the `gdb` debugger.
    (b) It doesn't work for function templates.
    (c) It only works if each header file has its own name.
    (d) `gcc`/`g++` doesnt support it.
    (e) It isnt part of the C++ standard.
    (f) It disables compiler warnings.
    (g) None of the above.

11. What is the purpose of header guards?

    (a) To allow templates to be instantiated.
    (b) To avoid redefinition errors.
    (c) To allow for faster compilation.
    (d) To ensure that a class's privacy is maintained.
    (e) None of the above

12. Which of the following files should be compiled?

    (a) Implementation Files
    (b) Header Files
    (c) Object Files
    (d) (a) and (b)
    (e) (a), (b), and (c)

13. What does the `-g` flag do when compiling with `g++`?

    (a) It causes the compiler to run faster (at the cost of more memory usage).
    (b) It does nothing.
    (c) It indicates that warnings should be enabled.
    (d) It specifies that the GNU compiler should be used.
    (e) It instructs the compiler to check for memory issues (like leaks).
    (f) None of the above.

14. What is the term for the place in one's code that a debugger is instructed to pause at?

    (a) stopline
    (b) pause-resume location
    (c) breakpoint
    (d) continue
    (e) None of the above

15. If a function wants to use arguments to "return" output values in addition to the regular `return` statement, what type modifiers should be used?

    (a) The arguments should be passed as non-const reference
    (b) The arguments should be passed as const reference
    (c) The arguments should be passed as a pointer
    (d) (a) and (b) both work
    (e) (a) and (c) both work
    (f) (b) and (c) both work
    (g) All of (a), (b), and (c) work
    (h) No type modifiers need to be used

16. What does the `-g` flag do when compiling with g++?

    (a) It allows for a **greater** level of optimization, meaning that the code should in principal run faster
    (b) It checks for adherence to the **google** style guide and warns about deviations
    (c) It switches to using the GNU compiler option so the code will be open source
    (d) It indicates that debugging information should be included in the executable
    (e) None of the above

17. Why are *using* declarations a bad idea in library headers?

    (a) It isn't bad, one should put their using declarations in the header.
    (b) Because every file including that header will get the declarations.
    (c) Using declarations increases redundancy as they much be included for each function that uses the library.
    (d) Because headers can't use namespaces.
    (e) Because the header should be explicit about its dependencies.
    (f) Because it is redundant, the implementation files will already have the declarations.

18. What is the cause of the following error message?

```
In file included from
cpp_unit_test:-12:
secret_messages/redact.cpp:5:8:
error: redefinition of
std::__cxx11::string
redact_all_chars(string const &)
```

    (a) The `redact_all_chars` function generates undefined behavior.
    (b) The `redact_all_chars` function tries to change its const reference parameter.
    (c) The header for the `redact_all_chars` function hasn't been included in the main file.
    (d) There is more than one implementation of the `redact_all_chars` function.
    (e) There is a type error in the `redact_all_chars` function.
    (f) None of the above.

19. When is it better to do a namespace merge (`using std::cout; cout << ...`) instead of stating the namespace for each use (`std::cout << ...`)?

    (a) When the namespace is `std`.
    (b) To allow the compiler to know what namespace a name comes from.
    (c) When you are writing a library for others to use.
    (d) When you will be using a name frequently.
    (e) You should always state the namespace (always do the latter option).
    (f) None of the above are true.

20. Why aren't header files compiled?

    (a) Because templates can't be compiled directly.
    (b) Because any file ending with .hpp (or .h) will be refused by the compiler.
    (c) Because doing so would expose private information.
    (d) Because header file guards prevent them from being compiled.
    (e) Because they are already included where needed by implementation files.
    (f) All of the above are true.

21. What does the `break` command do in `gdb`?

    (a) It sets a breakpoint.
    (b) It stops the program from accepting input.
    (c) It ends the innermost loop.
    (d) It causes the program to crash.
    (e) None of the above are correct.

22. What is the sole disadvantage of using the `#pragma` header guard instead of the traditional `#ifndef` guard?

    (a) It only works if every file has a different name.
    (b) It does not prevent multiple declarations when the header is included multiple times.
    (c) It only works if every header file exclusively uses it instead of the traditional guard.
    (d) It is not officially part of the language standard.

23. The `ls` command shows you the contents of the current working directory. Which of the following does it NOT show?

    (a) Executable files (like a.out)
    (b) Child directory names
    (c) Implementation files (like main.cpp)
    (d) Header files (like main.hpp)
    (e) It shows all of the above that are present

24. Which of the following is NOT a preprocessor statement?

    (a) `#define SOME_NAME`
    (b) `#include <iostream>`
    (c) `#pragma once`
    (d) `#ifndef SOME_NAME`
    (e) All of the above are preprocessor statements

25. How do you determine which line of your code is causing a segmentation fault?

    (a) The compiler warning will indicate the line that is causing the fault.
    (b) The memory checker (Valgrind) will indicate the line that is causing the fault.
    (c) The error message indicates the line that is causing the fault.
    (d) The runtime exception log will indicate the line that is causing the fault.
    (e) The debugger will halt on the line that is causing the fault.

26. What does this code output?

```
for (unsigned i = 4; i >= 0; --i) {
    std::cout << i;
}
```

    (a) `3210`
    (b) `4321`
    (c) `321`
    (d) `43210`
    (e) None of the above