**COP 3503**

Exam **1 Key**

**20 June 2014**

**Part I – Answer any seven (7) by marking the code or using the space provided. Strike through the questions you do not answer.**

1. (5) Is this program fragment legal? If not, how would you correct it? What should it print? Clearly indicate each character in the output string including white space such as new lines.

cout << "This is a long" ;

<< "string that takes" ;

<< "four lines to" ;

<< "print!" << std::endl;

No – it has a semicolon at the end of each line where it should only have one at the end of the last line – or – each output line should have cout at the beginning. (3 pts)

Also, each string begins and ends without white space, so the last word of one is run up against the first word of the next one – spaces should be added after long, takes, and to. When the first problems are fixed (but not the spaces inserted), it will output:

This is a longstring that takesfour lines toprint!

Give 2 points if they give this string (with the words run together), or if they insert spaces in their corrections and have them in the string as well, or if they insert them in the correction but show the string with the words run together. Give them only one point if they do not correct the string literals in the code and do not run the words together. Give 0/2 points if they do not give the output at all.

1. (5) Which of the following output statements are legal? For each, either explain why it is illegal or show what it will print.
   1. cout << "/\*"; - legal, outputs [/\*] - in string literal
   2. cout << "\*/"; - legal, outputs [\*/] - in string literal
   3. cout << /\* "\*/" \*/; - illegal – comment is /\* "\*/, leaving

the start of a string " \*/; with no end

* 1. cout << /\* "\*/" /\* "/\*" \*/; - legal, outputs [ /\* ] - comment /\* "\*/ followed by string " /\* "

followed by comment /\*" \*/

* 1. cout << int i = 5; - illegal – can't have a declaration

where the argument to stream operator should be

One point for each – needs to include explanation (0.5 points for explanation, 0.5 for legal+output or illegal)

Note that you can't just have int i on the line above and just cout << i = 5; either, even though i = 5 *is* an expression. To do this, you must have cout << (i = 5); instead.

1. (5) Is there anything wrong with the following function? If so, how would you fix it?

int firstCap(string s) {

for (int i = 0; i < n; ++i) {

if (isupper(s[i])) break;

}

return i;

}

There is plenty wrong! First, n is not declared or initialized at all (1 pt – 0.5 for each aspect), and the scope of i is only the for loop, so it cannot be accessed in the return statement (1 pt). To fix, you must declare and initialize n appropriately (2 pts - there are several right ways to do this, such as just have i < s.size() in the condition. Be careful, there are also wrong ways to do it, like using n = sizeof(s)). You must also declare i outside of the for loop (1 pt). (You should also catch case where there are no caps – but this is a logic error.)

A corrected version of the code (without a bool to catch the no uppercase letter case) follows.

int firstCap(string s) {

int i;

int n = s.size();

for (i = 0; i < n; ++i)

{

if (isupper(s[i]))

break;

}

return i;

}

1. (5) Is there anything wrong with the following function? If so, how would you fix it?

int numCaps(string new) {

\_Count = 0;

for (int i = 0; i < new.size(); ++i) {

if (isupper(new[i])) ++\_Count;

}

return \_Count;

}

Plenty of problems! First, new cannot be used as an identifier since it is a keyword in C++, and identifiers cannot start with an underscore followed by an uppercase letter, so \_Count will have to change also. \_Count is not declared either. (3 points)

To fix this, just change new and \_Count to valid identifiers, and declare the replacement for \_Count as int. (2 points)

1. (5) For each of the variables in the following code, indicate what their value is at the end of the code and state why.

int i; // default initialized to 0 since not in any fcn

bool b; // default initialized to false since not in any fcn

int main() {

int j; // undefined - not default initialized since in main()

int k = b; // initialized to 0 by type conversion from false

double d; // undefined - not default initialized since in main()

cout << i << " " << b << " " << j << " " << d << " " << k << endl;

return 0;

}

Answer for each given as comments in the code. 1 point each (including explanation 0.5 pts).

1. (5) What does the following code output with inputs 100.00 and 3? 100.00 and 2? 100.00 and 1? Why?

int main() {

int service;

double tab;

cout << "Enter tab: " << flush;

cin >> tab;

cout << "Enter service (1=OK, 2=good, 3=great): " << flush;

cin >> service;

double tip = 0.18 \* tab;

if (service > 1)

if (service > 2)

tip = tab \* 0.2;

else

tip = 0.15 \* tab;

cout << "Tip is " << tip << endl;

return 0;

}

Outputs are:

100 3 – 20

100 2 – 15

100 1 – 18

(1 pt each)

Not exactly what we wanted!

The problem is the dangling else statement, which attaches to the inner if statement. (2 pts)

if (service > 1)

{

if (service > 2)

{

tip = tab \* 0.2;

}

else

{

tip = 0.15 \* tab;

}

}

else {}

1. (5) Is there anything wrong with the following code? If so, how can you fix it?

int main() {

cout << "Enter amount of bill: " << flush;

cin >> double tab;

double total = tip = tab \* 0.18;

total += tab;

cout << "The tip is $" << tip;

cout << "For a total of $" << total << endl;

return 0;

}

You can't have a declaration double tab as the LHS for the >> stream operator (1 pt).

tip is not declared (1 pt). Fix by declaring double tab before the input statement (1 pt), getting rid of the double in the input statement, and declaring double tip before its use the double total = tip = tab \* 0.18; declaration. Note that it is OK to assign tip in the same statement that initializes total.

1. (5) Is the following code correct? If not, how would you correct it?

class Point {

int x\_pos;

int y\_pos;

public

Point(int x, int y) {

this->x\_pos = x;

this->y\_pos = y;

}

}

Point Point::sum(Point &p) {

Point sum = new Point(this->x\_pos + p->x\_pos),

this->y\_pos + p->y\_pos);

return sum;

}

The code should be corrected to include the proper type for the sum method and its declaration in the Point class, have sum as a pointer, not the object, insert missing : and ;, and either add a ( or remove a ).

class Point {

int x\_pos;

int y\_pos;

public: // 0.5 pt

Point(int x, int y) {

this->x\_pos = x;

this->y\_pos = y;

}

Point \*sum(Point&); // 2 pts – declaration, and correct form

}; // 0.5 pt

Point Point::sum(Point &p) {

Point \*sum = new Point((this->x\_pos + p->x\_pos), // 2 pts

this->y\_pos + p->y\_pos);

return sum;

}

1. (5) Is the following code correct? If not, how would you correct it? Only point out errors that will cause it not to compile, if any. What does (should) it do? Assume that sqrt() is a function that returns the square root of the value it is passed.

double sqrt(double);

class Point {

int x\_pos;

int y\_pos;

}

int Point::dist(&Point p) {

int i = p->x\_pos – x\_pos;

i \*= i;

int j = p->y\_pos – y\_pos;

j \*= j;

return sqrt(i + j);

}

Aside from the fact that there is no way to set the x\_pos and y\_pos values for a point since there are no mutator or constructor methods defined, which would force the member variables to be public if you wanted to change them (which will not cause a compiler complaint until you try to access them), there is a missing ; after the class declaration, the dist parameter list has the & misplaced, and there is no dist member declared in the class itself. Also, in dist, p is an object, and so dereference its members using '.' not '=>'. Note that the probably unintended cast of the double parameter and the double result of sqrt() to int is fine with the compiler, though at least the type of dist() would be better if it were double. It should compute the (L2 metric) distance between two points in a plane (1 pt).

double sqrt(double);

class Point {

int x\_pos;

int y\_pos;

public:

int dist(Point&); // 1 pt declare, 1 pt form

}; // 1 pt

int Point::dist(Point &p) { // 1 pt

int i = p.x\_pos – x\_pos; // 0.5 pt

i \*= i;

int j = p.y\_pos – y\_pos; // 0.5 pt

j \*= j;

return sqrt(i + j);

}

1. (5) Is the following code correct? If not, how would you correct it?

int main() {

int a[] = {1,2,3,4,5,6,7,8,9,10};

int i,j;

cout << "Enter two numbers between 1 and 10: " << flush;

cin >> i >> j;

aswap(i,j);

for (int \*b = a; b < sizeof(a); ++b)

cout << \*b << " ";

cout << endl;

return 0;

}

void aswap(int i, j) {

int tmp = a[i];

\*(a+i) = \*(a+j);

a[i] = a[j];

return;

}

aswap() must either be declared before main() or moved before main(). (1 pt) You can't compare a pointer to an integer in the for condition, and sizeof(a) is the wrong number for an index anyway – it would have to be sizeof(a)/sizeof(int) (1 pt). In aswap(),j is not given its type int (1 pt), and a is not accessible in that scope (1 pt) (have to make it global or pass a pointer in). A logical error is that a[j] should be assigned the value stored in tmp at the end of aswap()(1 pt). Another logical error is that the index numbers should be between 0 and 9, or really between 0 and sizeof(a)/sizeof(int)-1, and they are never checked.

void aswap(int i, int j) {

int tmp = a[i];

\*(a+i) = \*(a+j);

a[j] = tmp;

return;

}

int a[] = {1,2,3,4,5,6,7,8,9,10}; // make global or pass in to aswap

int main() {

int i,j;

...

int \*b = a;

for (int i=0; i < sizeof(a)/sizeof(int); ++i, ++b)

cout << \*b << " ";

cout << endl;

return 0;

}

**Part II – Answer any three (3) on three separate sheets of paper (one per problem)**

1. (5) What is the difference between a struct and a class? Why is this important? What programming habit can you develop that makes the difference unimportant?

The sole difference is that members are public by default in a struct while they are private by default in a class. This is significant because it means that the programmer has to consciously share members in a class, whereas by omission they can be shared in a struct. A good programming habit is always to label all members and to deliberately decide whether or not they should be public.

1. (5) What is the signature of a function? Why is it important?

The signature of a function is its name and the ordered list of types it takes for parameters. Note that the function return type is NOT part of its signature. The signature is important because it allows for static typing wherever the function is invoked (for parameters passed in); and it allows the compiler to determine which function or which version of a function is being called (parameter type list varies).

1. (5) What is the difference between a static and an automatic variable? Describe where they are allocated, their scope, and their lifetime.

A static variable is defined within some scope (typically a function) but is not allocated in the activation frame of the block in which it is declared, but rather in the data segment (or heap). Automatic variables only exist as long as the scope in which they are declared exists – they are allocated in the activation frame of the scope on the stack, rather than in the heap. Static variable endure until the program ends, but are not accessible from outside of their scope (unless a pointer to them is explicitly passed). In both cases, the variables are only accessible from within the scope in which they are declared or a nested scope that does not obscure them.

1. (5) What is separate compilation? Why is a header file required for separate compilation? What is a header guard, how does it work, and why is it used?

Separate compilation is the practice of compiling source code to machine code independently for each of two or more files that contain the code implementing a system. Header files contain declarations of classes, functions, variables, and constants that are needed for the various parts to interface properly. These allow the declarations to remain consistent across all components that use them. Without the header file, if the source to a separately compiled library is not available, then it would be difficult at best to get the declarations right to be able to use the facilities of the library. A header guard is a set of preprocessor directives that make sure that a given header file is included exactly once for each compilation that is performed – repeat declarations are not allowed by the compiler. It works by selectively including declarations from a header file only if a particular flag has not been set, and then setting that flag within the included declarations.

1. (5) What is a makefile? How do they work? Why are they used? Why are they important when using separate compilation?

A makefile is essentially instructions for the make program so that it can selectively execute sequences of commands depending on the relative ages of various files. The makefile allows the programmer to specify a target (usually a file) and the other files on which it depends. Only if the target is older than one or more of the files on which it depends will the target be “made”. If the target is to be made, then the command line commands that allow this to occur are located on lines immediately following the line with the target and its dependencies, with each line holding a command starting with a tab character. Makefiles allow not only for convenience when compiling a target, but they form documentation of how to do this (e.g., compiler flags and dependencies). When separate compilation is used, if the dependencies are captured correctly, then only the files that need to be updated are remade when a file changes. This saves time and space.

1. (5) What is the difference between creating an instance of class Foo using the declaration style (Foo f = Foo()) and using new? How does access to the object differ according to the instantiation method? When is it absolutely necessary to use new to instantiate an object?

If new is used, the object is allocated from the heap and persists until it is explicitly destroyed, whereas the declaration style instantiation can be used by automatic objects. New returns a pointer to the object, hence the pointer must be dereferenced before the members can be accessed, either using (\*objPtr) or the objPtr-> shorthand. It is absolutely necessary to use new to instantiate an object when it is instantiated in one scope but is intended for use in another, non-nested scope. In this case, the pointer value can be passed out of the scope and access to the object remains even when the scope has expired.