Zachary Thompson
12/12/14
CS 460 Final Exam

1. (10) a. What is structured programming? What are its benefits?

Structured programming involves the use of subroutines and control structures to organize code into conceptually logical blocks. The benefits are that code is easier to read and debug because each portion of code is responsible for a particular task, so you can think about the individual piece instead of the program as a whole.

b. What are the 3 standard categories of control structures? Give a C++ example of a structure for each of the 3 categories.

The three categories of control structures are selection statements (if/else, switch), iterative statements (while, for, do/while) and unconditional branches (subroutine calls).

```
int a = 0;
while (true) {
        if (a == -3) {
                return abs(a);
        }
        a -= 1;
}
```

while is the iterative statement, if is the selection statement, and abs is the unconditional branch.

2. (10) a. What is meant by "operator evaluation order"? Using C++ syntax, create and explain an example in which the order of operator evaluation affects the result.

Operator evaluation order determines which portion of a statement should be evaluated first. The acronym PEMDAS is an example of the order in which operators should be evaluated. An example where this matters is the following C++ statement:

```
int a = 2 + 3 * 2;
```

Since * has higher precedence than +, * gets evaluated first. Therefore the end result is a = 8, instead of a = 10 if + had been evaluated first.

b. What is meant by "operand evaluation order"? Using C++ syntax, create and explain an example in which the order of operand evaluation affects the result.

Zachary Thompson
12/12/14
CS 460 Final Exam

Operand evaluation order determines whether the left operand or right operand gets evaluated first. This is important because an operand can be an expression that alters the value of a variable used in another operand's expression.

```
int a = 5;
int b = ++a * (2 + a);
```

If the left operand of * is evaluated first, the result will be 6 * (2 + 6) and b will equal 48. If the right operand is evaluated first the result will be 6 * (2 + 5) and b will equal 42.

(10) The language C allows nested functions and uses static scoping to determine the appropriate scope of variables. What is the output of the following program? You may wish to show the intermediate values of the different variables for possible partial credit.

The output will be "a = 2; b = 3; c =7; d = 36; e =7". One of the areas of interest is inside function2 where a = 2 * c. Since a was not declared within the function, it uses the a from the outer function and changes its value from 2 to 6. b is set to the result of that function (which is a) so b and a both equal 6 and therefore the return value, which is assigned to d is 36. The second area of interest is inside function3 where c = a + b. Since c was not declared inside this function it uses the c from the outer function (main) and changes its value to a + b which is 7. e is set to the result of that function, which is once again 7. The original a and b defined in main do not get changed, since no inner functions overwrite them within main, and any time they are used in a function it is pass by value.

4. (10) Illustrate the stacks and queues used to evaluate the following expression written in our project language. What are the final values of a, b, and c if their initial values are 3, -1, and 2 respectively.
a = (b = b + c * 2) + 2 * (c = ++c += 1 << 10 >> 3);

Stack 1
a = 3, b = -1, c = 2
[a], [] => [a], [=]
Stack 2 is evaluated, b = 3 and b is pushed onto stack
[a, b], [=] => [a, b], [=, +] => [a, b, 2], [=, +] => [a, b, 2], [=, +] =>
[a, b, 2], [=, +, *]
Stack 3 is evaluated, c = 131 and c is pushed onto stack

[a, b, 2, c], [=, +, *] =>(2*c)=> [a, b, 262], [=, +] => (b+265) => [a, 265], [=] =>(a=265)=> [a] (complete)

Stack 2
a = 3, b = -1, c = 2
[b], [] => [b], [=] => [b, b], [=] => [b, b], [=, +] => [b, b, c], [=, +] => [b, b, c], [=, +, *] => [b, b, c, 2], [=, +, *] =>(c*2)=> [b, b, 4], [=, +] =>(b+4)=> [b, 3], [=] =>(b=3)=> [b] (result pushed onto stack 1)
Stack 3
a = 3, b = 3, c = 2
[c], [] => [c], [=] => [c], [=, ++] => [c, c], [=, ++] =>(c=3)=> [c, c], [=] => [c, c], [=, +=] => [c, c, 1], [=, +=] => [c, c, 1], [=, +=, <<] => [c, c, 1, 10], [=, +=, <<] =>(1<<10)=> [c, c, 1024], [=, +=, <<] => [c, c, 1024], [=, +=, >>] => [c, c, 1024, 3], [=, +=, >>] =>(1024 >> 3)=> [c, c, 128], [=, +=] =>(c+=128)=> [c, c], [=] =>(c=c)=> [c]
(result pushed onto stack 1)

End result: a = 265, b = 3, c = 131

5. (10) a. Given the following C data types and declarations, describe the use and storage requirements of each

Stype is a struct that has four different members. Each member occupies its own space in memory, so this struct can have separate values for all four members. The space requirement is sizeof(int) + sizeof(float) + sizeof(double) + sizeof(char) which is typically 4 + 4 + 8 + 1 = 17 bytes. The Utype is a union which has four value options. All four of the options occupy the same space in memory, so for all practical purposes only one actual value can be stored at a time. The amount of space required is the size of the largest type, which in this case is the double at 8 bytes.

b. How are unions related to the concepts of inheritance and polymorphism found in Object Oriented Programming?

Unions are related to inheritance and polymorphism because they provide flexibility. Like subclasses, based on the context unions can take on different values. This is related to polymorphism because a single union can provide functionality across multiple types just as a polymorphic function can service multiple types or classes. Also unions and inheritance-based classes both provide a mechanism to store multiple types within an array.

Zachary Thompson
12/12/14
CS 460 Final Exam

6. (10) Object Oriented Programming
a. What are the benefits and drawbacks of object oriented programming?

The benefit of object oriented programming is that it provides a a paradigm in which to model programs. It provides a mechanism for encapsulation and abstraction. This allows the programmer to separate the public interface from the implementation details. In the same way that a driver does not need to know the exact way in which an engine works yet is still able to drive a car, a programmer can use a class without needing to know the details of its implementation. Object oriented programming also provides a clean way to separate different needs of the program to different classes. Each class is concerned with its own tasks and in many cases doesn't need to know anything about any other classes in order to accomplish its goals.

The drawback of object oriented programming is that it requires a lot of function overhead. Function calls are the bread and butter of the OO paradigm. Another drawback is that the flexibility that inheritance provides, comes at a performance cost. All classes that use virtual methods must be allocated on the heap, which is a performance hit. Another drawback is that object oriented programming has become so pervasive that sometimes it is used in cases where another programming paradigm would make more sense.

b. What is function overriding?

Function overriding is a way for a subclass to provide an implementation for a method defined in the superclass. For example, a class called Shape might have a toString method that returns "I am a Shape". If a subclass called Square is created from Shape, it might want to override the toString method to say "I am a Square". Function overriding provides a way for Shape and Square to use the same method name to call to separate functions.

7. (10) a. How does a FORTRAN DO loop differ from a BASIC for loop?

The FORTRAN DO loop has a starting value, an ending value and an optional step. It is closed by END DO. The BASIC for loop has a starting value followed by the TO keyword and an ending value. At the end of the for loop the NEXT is called on the variable that the starting value was assigned to.

b. How does a Pascal for loop differ from a BASIC for loop?

PASCAL uses := for assignment. PASCAL is in a for <> to <> do format, while BASIC is in a FOR <> TO <> format.

c. How does a C/C++ for loop differ from a Pascal or BASIC for loop?

A C/C++ for loop has three parts: initialization, test, and code to run at the end of the loop (often an increment). C/C++ use curly braces to create blocks, while PASCAL and BASIC do not.

8. (10) What is the output of the following C++ loop. Why does it produce this output?

The output is either an infinite loop or a VERY long loop. The reason for this is once a >= 10 the second or clause kicks in. The second clause, a = a + b, will only ever evaluate to false if a+b is equal to 0. a and b will be positive numbers until they overflow, so the only way this loop will end is if at some point after they overflow a + b happens to equal 0. The beginning will look like this:
Before loop: 0 0
4 1
5 2
6 3
7 4
9 6
17 7
26 8
36 9

9. (10) Compilers and interpreters
a. What is meant by “single pass compilation”? How does C++ accommodate single pass compilation?

Single pass compilation means the compiler is able to read in the input file and create machine code on that first time through. C++ accommodates this by having forward declarations and doing lexical, syntactical and semantic analysis on the fly instead of requiring an entire pass for each one.

Zachary Thompson
12/12/14
CS 460 Final Exam

b. We developed a compiler for a small language in class. Part of the process included identifying "firsts" and "follows" for the non-terminal symbols in the LL(1) grammar for the language. What are the terminals in the "firsts" used for?

The terminals in the firsts are used to determine which rule we should go to next. If we are at the beginning of a line than some of the potential firsts are INTTYPE and DBLTYPE. If it is either of those, than we know that the current line is a declaration and can act accordingly, if the first is something else than we know the current line is a statement (or perhaps an error).

10. (10) Your boss has just returned from a conference, overflowing with enthusiasm about Yapl-Rapl! (Yet Another Programming Language to Replace All Programming Languages!) S/he has requested that you take the lead in learning and evaluating Yapl-Rapl!. Describe 5 activities you would employ to accomplish this task.

The five tasks I would evaluate are efficiency of the language, ease to write in the language, ease to debug in the language, community support for the language, and the documentation of the language. To evaluate the efficiency I would compare the runtime speeds of similar programs in a compiled language and an interpreted language. To evaluate the ease to write I would compare the time it took to write programs in the new language with a high level language and a low level language. The ease of debugging is a bit subjective, but some of the factors would include strong typing, error handling, and available environment debugging tools. Community support is very important because it represents the excitement for the new language. This support leads to packages and modules being created to extend the language, as well as tutorials and guides for beginners. The documentation is also very important to evaluate because it greatly contributes to the ease of programming in the language. Good documentation is paramount to the success and adoption of a language.