CS 280
Mid-term Exam
Review Examples and Exercises

Spring 2025

Mid-term Exam

- Lecture Topics included:
  - Lecture 1: Preliminaries (Ch.1)
  - Lectures 2-7: C++ Features
    - streams and files
    - Functions, pointers, references, and arrays
    - Structures & classes
    - Overloading, Generic Functions and Recursion
    - C++ Standard Template Library (STL)
  - Lecture 8: Description of Syntax (Ch. 3)
  - Lecture 9: Lexical analysis (Ch. 4.1-4.2)
  - Lectures 11 & 12: Variables, Bindings and Scopes (Ch. 5)
- Type of Questions Expected
  - Multiple-choice (20, 2 points each)
  - True/False (10, 1 point each)
  - Total points: 50

Preliminaries (Ch. 1): Examples

- What have been the strongest influences on programming language design over the past 70 years?
  a. Computer architecture only
  b. Programming design methodologies
  c. Both Computer architecture and Programming design methodologies
  d. Diversity of application areas

- What is the programming language category whose structure is dictated by the von Neumann computer architecture?
  a. Imperative
  b. Logic
  c. Functional
  d. Object oriented

C++ Features (Lectures 2-7): Example Questions

- Examine the following code segment and select from the following choices the missing expression to make this loop continue until either ten values have been read or the file runs out of data. Where, inData is assumed an input fstream object.

```
int count = 0;
inData >> value;
while  (___________________________){
  cout << value;
  inData >> value;
  count++;
}
```

```
a.  count < 10 && inData
b.  count < 10 || !inData
c.  count <= 10 && inData
d.  count < 10 && !inData
```

C++ Features: Example Questions

- Given the following struct definition for a Student, and the following declarations,

```
struct Student{
  int id;
  string name;
  float gpa;
};
Student astud;
Student * stptr = & astud;
```

```
Indicate which of the following choices is not a correct
method to access a Student field.
```

```
a. astud.name
b. (*stptr).name
c. stptr->name
d. *stptr->name
```

C++ Features: Example Questions

- Generic Functions
  - Given the following definition of a function template:

```
template <class T> T maximum(T x, T y) {
        T res;
        if(x > y) res = x;
        else res = y;
        return res;
}
```

Assume the following variables definitions in a program unit:

```
double x, y, z;
float i, j, k;
```

Which of the following calls to the maximum function from the program unit would cause a syntax error?

```
a. z = maximum(x, y);
b. k = maximum(i, j);
c. z = maximum(i, x);
d. z = maximum <double>(i, x);
```

C++ Features: Example Questions

- Given the definition of class One, and the MyFun partial function definition,

```
class One{
public:
One ( int x = 0) : val(x) { };
. . . . .
private:
int val;
};

void MyFun(){
One obj1(5), obj2;
int x = 4;
. . . .
obj2 = x * obj1;
. . .
}
```

```
Identify from the following
choices the correct declaration of
the operator*() function to be
added either as a member or as a
friend of the One class.

a. Member function of class One:
   One operator*(int val);
b. Global function: One
   operator*(int val, One obj);
c. Member function of class One:
   One operator*(int val, One obj);
d. Member function of class One:
   One operator*(One obj, int val);
```

- It is required that a binary operator * to be overloaded to allow the execution of the following assignment statement in MyFun() definition:

```
obj2 = x * obj1;
```

Describing Syntax Examples

- Given the following grammar with nonterminals *S*, *A*, and *B*?

  S → A a B b

  A → A b | b

  B → a B | a

  Which of the following sentences is a valid one in the language generated by this grammar.

  a. **bbbab**

  b. **bbaaaa**

  c. **Bbabb**

  d. **baab**

Describing Syntax Examples

- Given the following BNF grammar for a language with two infix operators represented by # and $.

  Foo ::= Bar $ Foo | Bar

  Bar ::= Bar # Baz | Baz

  Baz ::= x | y | (Foo)

a) Which operator has higher precedence: (i) $; (ii) #; (iii) neither; (iv) both
b) What is the associativity of the $ operator: (i) left; (ii) right; (iii) neither
c) What is the associativity of the # operator: (i) left; (ii) right; (iii) neither
d) Assuming that the start symbol is Foo, is this grammar: (i) ambiguous or (ii) unambiguous?
e) Give a parse tree for the following string: x $ x # y # (y $ x)

Describing Syntax Examples

- Given the sentence: x $ x # y # (y $ x)
- Derivation steps based on leftmost derivation

Foo => Bar $ Foo => Baz $ Foo => x $ Foo => x $ Foo => x $ Bar

⇒ x $ Bar # Baz => x $ Bar # Baz # Baz => x $ Baz # Baz # Baz = x $ x # Baz # Baz

⇒ x $ x # y # Baz => x $ x # y # (Foo) => x $ x # y # (Bar $ Foo)

⇒ x $ x # y # (Baz $ Foo) => x $ x # y # (y $ Foo) => x $ x # y # (y $ Foo)

⇒ x $ x # y # (y $ Bar) => x $ x # y # (y $ Baz) => **x $ x # y # (y $ x)**

- Repeat the derivation for the same sentence based on leftmost derivation by selecting an alternative RHS to show whether the grammar is ambiguous.

Describing Syntax Examples

- What is the correct EBNF for the following BNF rule?

```
term ::= term * factor
         | term / factor
         | term % factor
         | factor
```

a. term ::= term (* | / | %) factor

b. term ::= term {(* | / | %) factor}

c. term ::= factor {(* | / | %) factor}

d. term ::= factor (* | / | %) factor

- Look for recursion, use { } and combine with non-recursive RHS
- Look for common strings that can be factored out with grouping and options [ ].

Regular Expression Examples: Matching Strings to Regular Expressions

RegExpr	Meaning
x	a character x
\x	an escaped character, e.g., \n
M | N	M or N
M N	M followed by N
M*	zero or more occurrences of M
M+	One or more occurrences of M
M?	Zero or one occurrence of M
[*characters*]	choose from the characters in []
[aeiou]	the set of vowels
[0-9]	the set of digits
. (that's a dot)	Any single character

You can parenthesize items for clarity

Regular Expression Examples

- Regular Expressions: Creating expressions given a description.
  - A sequence of digits of any length that also contains the sequence of the five letters “hello” in that order, anywhere in the sequence.
    - **[0-9]*hello[0-9]***
  - A string that begins with a G and ends with an !, with one or more letters in between.
    - **G[a-zA-Z]+!**

Regular Expression Examples

- Regular Expressions: Matching strings to regular expressions
  - Given a regular expression and a string, indicate the character in the string where the mismatch happens:
  - [A-Z]+y?: “ENyce”
    - Solution: the lowercase letter ‘c’.
  - \-?[1-9][0]+: -5
    - The string ends before the pattern
  - ([A-Z][A-Z])+: GOOOOOGLE?
    - The ‘?’ in the string

Regular Expression Examples

- Which regular expression that matches a sequence of a string of zero or more even digits of any length followed by one or more letters?
  a. [0 2 4 6 8]+ [a-zA-Z]+
  b. [0 2 4 6 8][a-zA-Z]
  c. [0 2 4 6 8]*[a-zA-Z]+
  d. [0 2 4 6 8]?[a-zA-Z]*

Regular Expression Examples

- Given the following definition that describes regular expression of real numbers in scientific notation, what is the string of the following choices that does not match the given regular expression definition?

  [+-]? [0-9] \.? [0-9]* e[+-][0-9]+

  a. **1.25e+3**
  b. **0.25e-4**
  c. **.25e-2**
  d. **-2e+2**

Lexical Analysis

- A lexical analyzer is a pattern matcher for character strings
- A lexical analyzer is a "front-end" for the parser
- Identifies substrings of the source program that belong together - *lexemes*
    - Lexemes match a character pattern, which is associated with a lexical category called a *token*
        - `sum` is a lexeme; its token may be `IDENT`
- The lexical analyzer is usually a function that is called by the parser when it needs the next token.

Lexical Analysis Examples

- What does it mean when an automaton (FSA) representing a token pattern reaches the Final state?

  a. The token has been determined.
  b. There is an error in the input character string.
  c. The input character string is incomplete.
  d. The token has not been determined.

- Lexical analyzers are combined DFAs recognizing multiple patterns at once. (T/F)
- Lexical analyzers use table lookup to determine whether a possible identifier is in fact a reserved word or not. (T/F)

Lexical Analysis Example Questions

- What does a lexical analyzer generate?
  a. Lexemes.
  b. Tokens.
  c. Parse tree.
  d. Sentences

- Most existing languages can be tokenized using:
  a. Zero-character lookahead
  b. One-character lookahead.
  c. Teo-character lookahead
  d. Any-character lookahead

Variables, Bindings and Scopes: Example Questions

- The category of *Explicit heap-dynamic* variables are variables that are allocated and deallocated by explicit directives in C++ specified by the programmer. (T/F)

- In C++, variable declaration statements are not allowed to occur anywhere a statement can appear in a program. (T/F)

Variables, Bindings and Scopes

- Categories of Variables by Lifetimes
  - Static--bound to memory cells before execution begins and remains bound to the same memory cell throughout execution.
  - Stack-dynamic--Storage bindings are created for variables when their declaration statements are *elaborated*.
  - *Explicit heap-dynamic* -- Allocated and deallocated by explicit directives, specified by the programmer, which take effect during execution
  - *Implicit heap-dynamic*--Allocation and deallocation caused by assignment statements.
- The storage binding of all declared local variables in C++ functions are of the category of Stack-dynamic variables.

Variables, Bindings and Scopes

- Consider the following C++ code segment,

```
double z;
void myFun(void){
  int a;
  . . .
  for(int i = 0; i < 10; i++){
  . . .
  }
}
```

What are the lifetimes of the variables z, a, and i in the above code segment?

a. The life times of z and a are the program execution time, and the lifetime of i is the for-statement block execution time.
b. The life time of z is the program execution time, and the lifetimes of a and i are the function call execution time.
c. The life times of z and a are the program execution time, and the lifetime of i is the function call execution time.
d. The life time of z is the program execution time, the lifetime of a is the function call execution time, and the lifetime of i is the for-statement block execution time.

Variables, Bindings and Scopes

- Examples of categories of variables by lifetimes

```
int outside;    // external variable
                // exists for lifetime of the program
void myFunction() {
  static int counter=0;
  // exists for lifetime of the function call
  int val
  SomeClass x;
  SomeClass *p;

  // the memory returned by new comes from the heap
  // exists until cleaned up/freed (by user or by runtime
  // environment)
  p = new SomeClass();
}
```

  - The storage binding of all declared local variables in C++ functions are of the category of Stack-dynamic variables.

Variables, Bindings and Scopes

- Given the following declaration,

```
int *ptr;
int x = 5;
```

  - What is the *r-value* of *ptr* in the following statement?

```
ptr = & x;
```

a. R-value of x variable.
b. Pointer variables do not have an r-value.
c. L-value of x variable.
d. R-value for ptr has not been defined

Variables, Bindings and Scopes

- Given the following declaration in a C++ function:

  ```
  int * ptr = new int [5];
  ```

  The `ptr` variable is defined and initialized to:

  a. The address of an array stored in the run-time stack Segment.
  b. The address of an array stored in the heap segment.
  c. The address of an array stored in the data segment.
  d. The address of an array stored in the code segment.

Variables, Bindings and Scopes

- **Scope**
  - Example: Consider in C/C++ Nested blocks

```
void sub() {
 int count;
 while (...) {
 int count;
   count++;
   ...
 }
 …
}
```

  - The reuse of a variable name is allowed in C++, but is not in Java.

Variables, Bindings and Scopes

- Scope (cont'd)
  - Example: Consider the following C++ function definition,

```
void function(void){
 int a, b, c;//definition 1
 . . .

 while (. . .) {
   int b, d;//definition 2
   . . .
   . . .
   if (. . .) {
        int e, a;//definition 3
        . . . //Point 1
   }
        }
}
```

a. Variables a and e from definition 3, variables b and d from definition 2, and variables c from definition 1.
b. Variables a and e from definition 3, and variables b and d from definition 2.
c. Variables a and e from definition 3, and variables b and c from definition 1.
d. Variables a and e from definition 3 only.

Determine the visible variables at Point 1, inside the if statement of the function, using the labelled definition statements by comments in the function,