1 Notes and disclaimer

These review materials only cover topics discussed after the midterm. For a review of pre-midterm content, see the midterm review materials. These questions are intended to be a selection of topics which may be tested on the final, but have been written without knowledge of the specific contents of the final exam and thus may emphasize different areas. The number of questions in a category is not representative of the representation of that category on the final exam.

2 Context-free Languages

1. Give an example of a language which is context-free but not regular.

2. What does it mean for a context-free grammar to be ambiguous? Give an example of an ambiguous grammar.

3. Provide both a leftmost and rightmost derivation of $aaabbc$ using the grammar specified by the following production rules:

   $$ S \rightarrow aABc $$
   $$ A \rightarrow aA $$
   $$ A \rightarrow \epsilon $$
   $$ B \rightarrow bb $$

4. Design a context-free grammar for the language of strings over $\Sigma = \{a,b,c\}$ of words of the form $a^n b^{n+m} c^m$ with $n, m > 0$.

5. Recall that the WLM rule for if statements looks like this (with some words abbreviated):

   $$ \text{stmt} \rightarrow \text{IF LPAREN test RPAREN LBRACE stmts RBRACE ELSE LBRACE stmts RBRACE} $$

   Add grammar rules to support the following extended if statement syntax:

   if (condition1) {
       // statements
   } else if (condition2) {
       // more statements
       // [more else ifs]
   } else { // else is still mandatory
       // statements
   }
That is, you should support any number (zero or more) “else if”s in a WLM if statement.

3 LL Parsing

3.1 Identifying incorrect grammars

Argue why each of the following CFGs are not LL(1) by constructing enough of the predictor table to show an issue:

1. 

\[ S' \rightarrow \cdot \cdot S \cdot \cdot \]
\[ S \rightarrow Sab \]
\[ S \rightarrow xy \]

2. 

\[ S' \rightarrow \cdot \cdot S \cdot \cdot \]
\[ S \rightarrow abc \]
\[ S \rightarrow abd \]
\[ S \rightarrow bSc \]

3.2 Parsing correct grammars

Construct a predictor table for the following CFG and use it to parse the string \( \cdot deccb \cdot \).

\[ S' \rightarrow \cdot \cdot S \cdot \cdot \]
\[ S \rightarrow aY \]
\[ S \rightarrow Zb \]
\[ Z \rightarrow XY \]
\[ X \rightarrow de \]
\[ X \rightarrow \epsilon \]
\[ Y \rightarrow ccY \]
\[ Y \rightarrow \epsilon \]

4 LR Parsing

4.1 Identifying incorrect grammars

Show each of the following grammars are not LR(0) by constructing enough of the automaton to show an issue:

1. 

\[ S \rightarrow aSb \] \hspace{1cm} (0)
\[ S \rightarrow a \] \hspace{1cm} (1)
2.

\[ S \rightarrow Xab \quad (0) \]
\[ S \rightarrow Ycd \quad (1) \]
\[ X \rightarrow e \quad (2) \]
\[ Y \rightarrow e \quad (3) \]

4.2 Parsing correct grammars

Construct a SLR(1) machine for the following CFG and use it to parse the string \( \overrightarrow{baabb} \overleftarrow{b} \).

\[ S' \rightarrow \overrightarrow{X} \quad (0) \]
\[ X \rightarrow XbAb \quad (1) \]
\[ X \rightarrow \epsilon \quad (2) \]
\[ A \rightarrow Aa \quad (3) \]
\[ A \rightarrow \epsilon \quad (4) \]

5 Errors in WLM

Each of the following WLM programs contains one or more errors. For each error you’re given a hint about roughly where the error occurs via an underline. For each program, identify:

- Which error would be detected first by the compiler (you may assume that if two errors occur in the same phase, the one which appears first in the code occurs first).
- Which phase of compilation that error occurs in.
- A brief description of why it is an error.

1. \( \text{int wain(int a, int b, int c) \{} \)
   \[ \text{a = a + 1;} \]
   \[ \text{int d = 3;} \]
   \[ \text{return b;} \]
   \[ \} \]

2. \( \text{int wain(int a) \{} \)
   \[ \text{return a * a;} \]
   \[ \} \]

3. \( \text{int wain(int a, int b) \{} \)
   \[ \text{if (a < b) \{} \]
   \[ \text{return a;} \]
   \[ \} \]
   \[ \text{return a > 3 ? a : b;} \]
   \[ \} \]
4. int wain(int a, int b) {
    int a = 0;
    return a + b;
}

5. int f(int a) {
    while (a < b) a = a + 1;
    return a;
}

   int wain(int a, int b) {
    return f(a, b);
}

6 Compiling code

Alice is frustrated with the lack of switch statements in WLM and decides to add them to the language, using the same syntax as C. Which parts of the compiler (scanning, parsing, context-sensitive analysis, and code generation) need to be changed? For each part that needs to be changed, give a high-level overview of what needs to be added, removed, or otherwise changed in that stage.

For a reminder of C’s switch statement syntax, Alice might want to write the following code:

```
switch (3 + 4) {
    case 5: foo(); break;
    case 7: bar(); break;
    default: break;
}
```

7 Code Generation

7.1 Translating WLM programs to MIPS

Give MIPS assembly code which are equivalent to the following WLM code snippets. For full marks, your code should work even when the statements, expressions, and mathematical operations within the main statement or expression are replaced by arbitrary other subexpressions, statements, or operations. You may use the following WLM conventions in addition to the usual MIPS conventions:

- $29$ is the frame pointer.
- $1$ and $2$ contain the arguments to a procedure. $\text{Fproc}$ is the label associated with procedure $\text{proc}$.
- $\text{ST}[a]$ is the offset of $a$ from $29$.
- $3$ contains the result of any expression or procedure call.
- $4$ contains 4, and $11$ contains 1.

You may also reuse the code for previous parts by saying, for example, \texttt{code(part1)} to refer to the code generated by part 1. You may do this even if you didn’t complete the previous part.
1. 4
2. 4 + 4
3. ((4 + 4))
4. a
5. a = ((4 + 4));
6. while (a < 4) {
   a = ((4 + 4));
}
7. f(g(), 4)

## 7.2 Generating code for new rules

Bob is maintaining a WLM compiler and his users are frustrated that they have to type “i = i + 1”, “c = c + d”, and other cumbersome expressions: they miss the “+=” syntax from C. As a result, he modifies the scanner so that it can generate a “PLUSEQ” token for +=, and adds the following rule to the grammar:

```
statement → ID PLUSEQ expr SEMI
```

However, he’s forgotten MIPS and is stuck on code generation. Write Scala, C++ or Racket code to generate the MIPS assembly code corresponding to this rule and print it to standard output. You may use `symbolTable(ID)` to refer to the offset of the variable from $29$, and `code(expr)` to refer to the code corresponding to `expr`.

## 8 Memory Management

### 8.1 Fixed-size memory management

Finish the WLM program below which implements a fixed-size memory allocator for `triple`. You may assume some of the functions (specified below) are implemented already, and that WLM has been extended to support three-parameter functions. You need to implement `triple` and `detriple`.

```c
// Implement the triple() and detriple() functions below
// using the allp1 library as well as the helpers defined below.

int init () {
   // You may assume this function is defined correctly
   // to initialize the arena if it isn’t already initialized,
   // and do nothing otherwise
}

int freelistFirst () {
   // You may assume this function is defined correctly.
   // and it returns the address of the first element
   // in the freelist. The freelist will always be nonempty unless
```
// no memory is available, in which case this function returns 0.
// All elements in the freelist point to non-overlapping blocks
// of size 12. The word pointed to by an element of the freelist
// is the next element of the freelist.
}

int setFreelistFirst(int pointer) {
    // You may assume this function is defined correctly
    // and it sets the top of the freelist to the pointer parameter.
    //
    // NOTE: This will replace the current freelist in its entirety!
}

int triple(int a, int b, int c) {
    // If 12 bytes of memory are available, this function
    // should store a, b, and c in that memory and return
    // its address. Otherwise it should return 0.
}

int detriple(int p) {
    // This function should make the memory associated
    // with p available for reuse.
}

As an additional exercise, try implementing the other functions.

8.2 Variable-sized memory management

In both cases, the unspecified implementation of malloc should be $O(1)$ for sparse arenas, and free should always be $O(1)$.

1. Suppose a user requests 4 bytes of memory. Give a memory diagram for a block of memory, including the size of each component, that might be returned by malloc. Indicate where the pointer returned by malloc points to within this block with an arrow.

2. Suppose a user frees the block of memory allocated in the previous part. Assuming the block is not merged with any adjacent blocks, give a memory diagram for the free block of memory, including the size of each component.

3. Explain how your block design in the previous parts allows free blocks to be merged with adjacent blocks when necessary in $O(1)$ total time.

8.3 Different fit strategies

Suppose we use a much simpler block layout that in the previous question (which may not support efficient frees), where each block has at least eight bytes in it and free blocks simply have their size in bytes followed by a pointer to the next block, while allocated blocks simply have their size in bytes followed by the user’s memory. Show that best fit is neither always better or worse than first fit by giving (and explaining via a sequence of arena diagrams), assuming that the head of the freelist is always the most recently deallocated node:

1. A sequence of allocations and deallocations which can be satisfied by best fit but not first fit.
2. A sequence of allocations and deallocations which can be satisfied by first fit but not best fit.

Use a 256-byte arena in both cases.