

## Summer 2008 Final Exam

---

Your Name: \_\_\_\_\_

SUNet ID: \_\_\_\_\_

### Instructions:

- **Honor.** You must complete this exam alone and in the time specified. You must follow the Stanford Honor Code.
- **Time Limit.** You have from 4:00pmPDT Friday August 15th to 4:00pmPDT Saturday August 16th 2008 to complete this exam.
- **Show and explain your work.** Do this whenever possible. It allows us to better characterize your understanding of the material - and better grade your hard work.
- **Notes.** You may refer to any books, notes, or other printed material. **You must *not* use any electronic materials or equipment (e.g. the Internet or dcc).**
- **Submission for All Students.** Submit your work **via e-mail** to `dgu@stanford.edu`. Also, Ian will be outside the Bytes cafe (across from the Gates building) at 4:00pmPDT on Saturday if you wish to turn in your exam in person. You must submit your work by the 4:00pmPDT deadline.
- **Questions During the Exam.** Feel free to e-mail both of us if you have questions. Responses will not be immediate, so please be patient!

Problem #	Points Earned	Points Possible
1		5
2		10
3		20
4		30
5		15
6		20
7		25
8		30
<b>Total</b>		155

**Problem 1 Hard Hat Area**

Construct an equivalent NDFFA from the following regular expression over the alphabet  $\{a, b\}$ :

$(aaa \mid b)^*$

**Problem 2 Nothing Ever Changes**

Minimize the DFA shown in Figure 1. Show your work step-by-step and draw the resulting DFA.

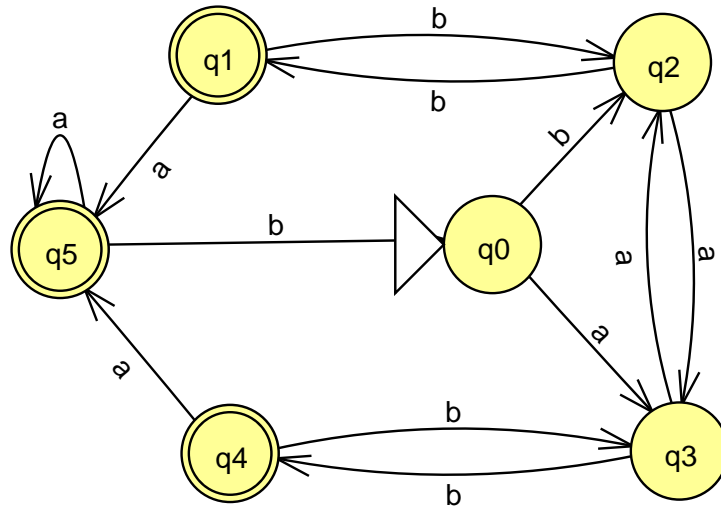


Figure 1: The DFA to minimize (Problem 2).

### Problem 3 Heads Up

Write the TAC code for *only* the `drawLinePoints` method in following Decaf source excerpt. Also construct a vtable for the `Vector2i` methods which you need to call. Space to write your answer is provided on the next page.

```
void drawLinePoints( Vector2i a, Vector2i b, int distBetweenPoints ) {
    Vector2i dir;
    Vector2i unitDir;
    Vector2i vNextPoint;
    boolean movingAway;
    int distSquaredToBPrev;
    int x;
    int y;

    // determine vector to take us from "a" to "b"
    dir = b.subtract(a);
    unitDir = dir.makeUnitVector();
    vNextPoint = unitDir.multiply(distBetweenPoints);

    // loop until we are completely on the other side of "b"
    x = a.getX();
    y = a.getY();
    movingAway = false;
    distSquaredToBPrev = 0x8FFFFFFF;
    do {
        int distSquaredToB;
        int dx;
        int dy;

        FillPixel(x, y); // FillPixel is a built-in which takes two ints
        x += vNextPoint.getX();
        y += vNextPoint.getY();

        // check to see how far we are from our destination now
        dx = b.getX() - x;
        dy = b.getY() - y;
        distSquaredToB = dx*dx + dy*dy;

        // determine whether the next mov would make us closer or not
        movingAway = distSquaredToB > distSquaredToBPrev;
        distSquaredToBPrev = distSquaredToB;
    }
    while( !movingAway );
}

class Vector2i {
    int x;
    int y;

    // ...
}
```

Figure 2: Generate the TAC for this code (Problem 3).

This page is blank to provide you with space to write your answer for Problem 3.

## Problem 4 The Road Not Taken

### 4.1 To Choose or Not to Choose?

Compare and contrast the advantages, disadvantages, and implementation costs of making all methods dynamically dispatched as in Decaf, versus only specifically indicated methods in C++ (with the `virtual` keyword).

### 4.2 Varying Variable Sizes

Discuss what changes you would have to make to your compiler to support variables of sizes other than 32 bits (e.g. 64 bit long longs, 16 bit shorts, and bit fields). Consider only the code generation phase of the compiler.

### 4.3 Hello World

In a language like Python, the methods defined by a particular instantiation of an object may be changed at runtime. Discuss the high-level changes that would need to be made to your compiler in order to support this form of dynamic typing. Be sure to discuss each phase of the compiler. A simple example of Decaf source code taking advantage of this functionality is shown below.

```
class Hello {
    void print() {
        Print("hello ");
    }
}

void printWorld() {
    Print("world\n");
}

void main() {
    Hello o;

    o = New(Hello);
    o.print();
    o.print = printWorld;
    o.print();
}

Output of this code:
hello world
```

Figure 3: Discuss the high-level changes that would be needed in order to support this code (Problem 4.3).

## Problem 5 Context Free Grammars

### 5.1 Grammar writing

Write a context-free grammar for the language of strings that consist of a series of zeroes followed by a series of the same number of 1s. Examples include:

- (the empty string)
- 01
- 0011
- 000111

### 5.2 Advanced grammar writing

Write a context-free grammar for the language of strings that consist of an equal number of zeroes and ones. Examples include:

- (the empty string)
- 01
- 10
- 0101
- 1100
- 0011
- 1010

### 5.3 Grammar debugging

The Decaf specification states that the assignment operator should not chain. Several people, including your instructors, attempted to implement this using a precedence directive in yacc:

```
%nonassoc T_Equal
```

However, using the grammar from the Decaf specification results in the string

```
a = b = c
```

being accepted. Why does the nonassoc precedence directive not cause the string to be rejected? (Don't worry, we still gave you points if your assignment did not match the specification on this point)

### Problem 6 Top-Down Parsing

This question uses the following grammar:

$$A \rightarrow a \mid BaE \mid Ca \mid Da$$
$$B \rightarrow bb \mid cb \mid \epsilon$$
$$C \rightarrow Ad$$
$$D \rightarrow bc$$
$$E \rightarrow aEa \mid \epsilon$$

#### 6.1 LL(1) parse table generation

Fill in the parse table for this grammar on the following page. If conflicts occur, indicate in which cells this happens by writing multiple productions in that cell. Circle cells that contain conflicts.

**6.2 Grammar revision**

Rewrite this grammar to be LL(1) without changing the language that it accepts.

A	a	b	c	d	\$
B					
C					
D					
E					

**Problem 7 Bottom-up parsing**

This question uses the following grammar:

$$S \rightarrow [A] \mid (B) \mid [B] \mid (A)$$
$$A \rightarrow a \mid A+A$$
$$B \rightarrow a \mid B*B$$

(Note that the nonterminals A and B both produce the terminal "a". There is no terminal "b".)

**7.1 LR(1) parsing**

Generate the configurating sets for an LR(1) parser. Indicate any conflicts, and explain how you would resolve them in bison. Explain whether the conflicts are shift-reduce or reduce-reduce. If you encounter a state with conflicts, you do not need to draw the states that would be reached by shifting more input *from the conflicted states*.

**7.2 LALR(1) parsing**

Show what changes would have to be made to update these configurating sets to make an LALR(1) parser. You do not need to copy over sets that do not change, but please indicate which states from the original parser will be removed, and which states were the source for any new states. Indicate any conflicts that result. Explain whether the conflicts are shift-reduce or reduce-reduce.



c. What changes would need to be made to the semantic analysis phase of the compiler in order to enable this feature?

d. What changes would need to be made to the code emitted by the compiler to enable this feature?

## 8.2 Object destructors

Consider adding object destructors to Decaf.

a. Can the scope manager safely call an object's destructor when it goes out of scope?

b. Does the runtime environment need to become aware of scopes in order to implement the new change?

- c. Do your answers to 8.2a and 8.2b change if we modify Decaf to allow allocation of objects on the stack rather than the heap? Explain.

### 8.3 Dynamic scoping

Consider changing Decaf to allow some limited dynamic scoping. Specifically, consider adding the feature that if a variable name is not found in the current scope, then the scope of the calling function is checked for that variable name. For example, the following program would result in “5” being printed.

```
void foo()
{
    x = 5;
}

void main()
{
    int x;
    x = 3;
    foo();
    Print(x);
}
```

**Output of this code:**  
5

Figure 5: Example of code using dynamic scoping (8.3).

- a. Give examples of some features that would have to be removed from compile-time semantic analysis.

