UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE AND ENGINEERING

CSC467– Compilers and Interpreters
Final Examination, Dec. 14, 2018

Exam Type: A, Calculator Type: 4

Duration: 2 and ½ hours

Examiner: Xu Zhao

Name: ______________________________

UTOR ID: ______________________________

Student Number: ______________________________

This exam contains 12 pages (including this cover page) and 6 questions. Total of points is 100. Good luck!

Distribution of Marks

Question	Points	Score
1	10	
2	25	
3	10	
4	10	
5	20	
6	25	
Total:	100	

1. (10 points) **True or False.** Clearly mark True or False for the following statements.

1) True False Bottom-up parsing traces a right-most derivation in reverse.

2) True False SLR parsing can resolve shift-reduce conflicts for all LR(0) automatas.

3) True False The variable liveness analysis result is used in register allocation.

4) True False In C++, each object contains a `this` pointer in its memory structure.

5) True False By default, Java uses lexical scoping, and C uses dynamic scoping.

6) True False A program's `.text` segment stores the global variables.

7) True False Semantic analysis can determine whether a program has infinite loops.

8) True False Bottom-up parsing requires a non-left-recursive grammar.

9) True False If `class A` inherits `class B`, `B a = A();` will cause object slicing.

10) True False A `unique_ptr` uses reference counting to manage dynamic memory.

2. (25 points) **Bottom-up Parsing** Consider the following augmented context free grammar. Non-terminals are $\{S', S, A\}$, terminals are $\{a, b\}$.

$$S' \to S \qquad (0)$$
$$S \to A\ A \qquad (1)$$
$$A \to a\ A \qquad (2)$$
$$A \to b \qquad (3)$$

a) (10 points) Draw the LR(0) automata for the bottom-up parsing. Use I_n to name each state. The initial state is I_0.

b) (10 points) Complete the following SLR parsing table. You do not need to fill in all the rows.

State	ACTION			GOTO	
	a	b	$	S	A
I_0					

Part c) is on the next page.

c) (5 points) Parse the string `aabab`. Write down the **reverse** of a right-most derivation (**NOT** the parse tree). Specify the grammar rule you are using to reduce on each step.

3. (10 points) **Program Analysis.** Perform the **Andersen-style** pointer analysis on the following program.

```
p = &a;
p = &b;
q = &c;
*q = p;
```

Draw the pointer reference graph.

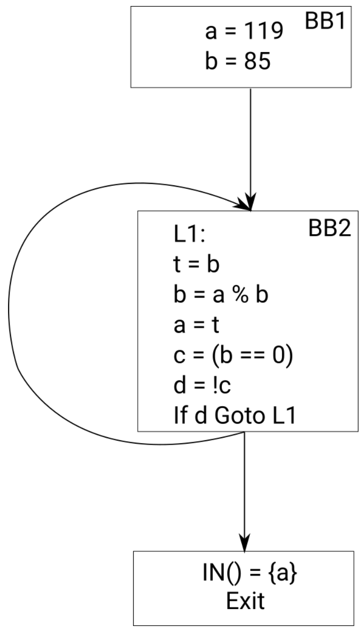
4. (10 points) **Instruction scheduling.** Consider the following Three Address Code:

```
1    b = 3
2    a = b + 1
3    c = b + 4
4    b = b + 5
5    a = 2 + b
6    c = 5
7    d = a + b
8    b = 10
```

a) (5 points) Suppose every instruction can finish in 1 cycle and we have infinite hardware resources. Write down the optimal instruction scheduling that requires the minimum number of cycles.
Note: Instructions with data dependencies must be placed in different cycles.

b) (5 points) Suppose every instruction can finish in 1 cycle and we can at most execute 2 instructions in parallel in one cycle due to hardware limitation. Write down the optimal instruction scheduling that requires the minimum number of cycles.
Note: Instructions with data dependencies must be placed in different cycles.

5. (20 points) **Register allocation.** Consider the following control flow graph. In the TAC code, `d = !c` is the boolean `NOT` operation, and `b = a % b` is the modulo operation. The live variable set before the `Exit` basic block contains a single variable `{a}`.



a) (10 points) Infer the live variable set before (`IN()`) and after (`OUT()`) each instruction.

Instruction	IN()	OUT()
a = 119		
b = 85		
L1:		
t = b		
b = a % b		
a = t		
c = (b == 0)		
d = !c		
If d Goto L1		

Part b) and c) of the question are on the next page.

b) (5 points) Draw the register interference graph that contains all the temporary variables `{a, b, c, d, t}`.

c) (5 points) Show the minimum number of registers needed for the variables by coloring the register interference graph. Prove that your solution is optimal.

6. (25 points) **Runtime environment and optimization.** Consider the following 64-bit **x86** assembly. By convention, the function return value is stored in the register **%eax**.

```
.globl    myfunction
myfunction:
pushq    %rbp
movq    %rsp, %rbp
movl    $872, -20(%rbp)
movl    $721, -16(%rbp)
movl    -16(%rbp), %eax
movl    -20(%rbp), %edx
andl    %edx, %eax
movl    %eax, -12(%rbp)
movl    -20(%rbp), %eax
movl    %eax, -8(%rbp)
movl    -16(%rbp), %eax
movl    -8(%rbp), %edx
andl    %edx, %eax
movl    %eax, -4(%rbp)
movl    -4(%rbp), %eax
popq    %rbp
ret
```

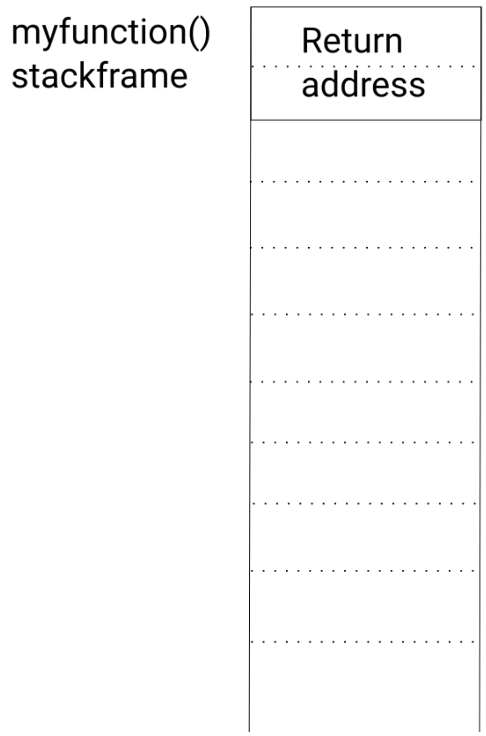
We provide the following table of explaining the **x86** assembly code semantics.

Assembly	C Semantic
`ret`	`return;`
`movl $872, -20(%ebp)`	`int *p = %ebp - 20; *p = 872;`
`movl -20(%ebp), %eax`	`int *p = %ebp - 20; %eax = *p;`
`andl %edx, %eax`	`%eax = %eax & %edx;`

Table 1: AT&T x86 assembly semantics

Answer the questions on the next page.

a) (10 points) Complete the following figure that describes the stack frame when the code is executing on line **8**. Suppose each cell is 4 bytes. Pointers or register values may occupy more than 1 cell (e.g. the return address is 8 bytes, therefore it occupies 2 cells). Only fill in the cells that are needed, and you are free to draw more cells if necessary.



b) (10 points) You are asked to optimize the above assembly by eliminating as many code lines as possible. Write down the **line numbers** of the assembly code that can be safely removed.

Note: you are only allowed to **remove** the assembly code. For example, if you are going to remove the code on line 1, 3, 7, 8, 9, write down "1, 3, 7-9".

c) (5 points) Write down the **simplest** C code that is equivalent to the optimized assembly code.

This page is intentionally left blank to accommodate work that wouldn't fit elsewhere and/or scratch work.

This page is intentionally left blank to accommodate work that wouldn't fit elsewhere and/or scratch work.