Arizona State University

Computer Science and Engineering

CSE 340 – Spring 2017

Homework 3

**Due Friday 17 March 2017**

**I Parsing.** Consider the following grammar.

S ➝ A C

S ➝ B D

A ➝ a A

A ➝ ɛ

B ➝ a B

B ➝ a

C ➝ c

D ➝ d

Clearly this grammar does not have a predictive recursive descent parser because,

FIRST(A C) ∩ FIRST(B D) ≠ ∅

Nonetheless, it is possible to have a recursive descent parser with backtracking for the grammar. **You are asked to write a recursive descent parser with backtracking for the grammar.**

Your parser should be a recursive descent parser which means that for every non-terminal, you should have a parsing function that parses the part of the input that corresponds to the non-terminal starting from a given point.

Every parsing function for a non-terminal should return either true or false. If the function returns true, then it should consume the part of the input corresponding to the non-terminal. If the function returns false, then it should leave the input undisturbed by un-getting all the tokens that it might have consumed.

If you write a code that reads all tokens upfront to determine which of the two options A C or B D to take, you will not get any credit for this problem.

**II Static and Dynamic Scopes.** Consider the following program written in the C syntax.

int a, x, y, z;

void print(int c){

if (c == 0)

printf(“g: %d, %d, %d\n”, a, x, y, z);

else

printf(“main: %d, %d, %d\n”, a, x, y, z);

}

int f(int x, int y)

{

int a, b, c;

a = 1;

b = 2;

c = 3;

x = g(10);

{

int y, z;

y = g(x); **/\* point 1 \*/**

z = g(1); **/\* point 2 \*/**

printf(“f1: %d, %d, %d\n”, a, x, y, z); **/\* point 3 \*/**

}

printf(“f2: %d, %d, %d\n”, b, x, y, z); **/\* point 4 \*/**

return x \* a – y \* b + z \* c;

}

int g(int a)

{

print(0); **/\* point 5 \*/**

return (x + y - z) \* a;

}

int main()

{

int a = 2;

x = 3;

y = 12;

z = f(x, y);

z = g(a); **/\* point 6 \*/**

print(1); **/\* point 7 \*/**

return 0;

}

**What is the output of this program? You should show the output of every printf() on a separate line.**

**You do not have to show your work, but if your answer for some lines are not correct, you will not get partial credit for those line if you do not show your work.**

Global:

* a =
* x = 3
* y = 12
* z = ~~126~~ -222

Local:

* x = ~~3~~ ~~150~~
* y
  + f: y = ~~12~~
  + braces in f: y = ~~2250~~
* a
  + f: a = ~~1~~
  + g: a = ~~10~~ ~~150~~ ~~1~~ 2
* b = 2
* c
  + f: c = ~~3~~
  + print: c = ~~0~~ ~~0~~ ~~0~~ ~~0~~ 1
* z = ~~15~~

**Output:**

g: 0, 3, 12 //point 5  
g: 0, 3, 12 //point 5 via point 1  
g: 0, 3, 12 //point 5 via point 2  
f1: 1, 150, 2250 //point 3  
f2: 2, 150, 12 //point 4  
g: 0, 3, 12 //point 5 via point 6  
main: 0, 3, 12

**III Pointer semantics.**

struct T {

int data;

struct T\* next;

};

struct T\*\* p;

struct T\*\* q;

struct T\*\* r;

struct T\* c,d;

int main() {

struct T\*\* a;

{ struct T\* b;

a = (struct T\*\*) malloc(sizeof(struct T \*)); // memory 1

\*a = (struct T\*) malloc(sizeof(struct T)); // memory 2

b = (struct T\*) malloc(sizeof(struct T)); // memory 3

(\*b).next = \*a;

p = &b;

b = (struct T\*) malloc(sizeof(struct T)); // memory 4

(\*(\*a)).next = b;

**// point 1**

c = (struct T\*) malloc(sizeof(struct T)); // memory 5

p = a;

a = c;

b = (struct T\*) malloc(sizeof(struct T)); // memory 6

**// point 2**

q = &b;

r = &c;

a = (struct T\*) malloc(sizeof(struct T)); // memory 7

(\*b).next = \*a;

d = \*a;

**// point 3**

free(a);

free(b);

**// point 4**

}

**// point 5**

}

**Draw a box-circle diagram for all visible variables at point 4. Also include in the diagram all memory locations that are garbage and the dangling references. Do the same for point 5.**