Sabancı University
Faculty of Engineering and Natural Sciences
CS305 – Programming Languages
QUIZ I

April 20, 2020

PLEASE NOTE:

- There are 3 questions in this exam.
- Provide only the requested information and nothing more.
- Unreadable, unintelligible and irrelevant answers will not be considered.

Question 1) [**5 points**] Your friend comes to you with this wonderful idea that he thinks would put an end to this ongoing battle of programming languages. Your friend wants to design a programming language that can be used for any purpose, in any domain.

What would your response be to your friend? Would you agree (in this case please briefly give us some high level design features of this all–purpose programming language), or would you disagree (in this case please explain why you disagree)?

Question 2) [**5 points**] Consider a context free grammar G that can generate a sequence of numbers `+1` and `-1` with the following properties:

- If we add the numbers in a sentence in the language of G, they sum upto zero.
- If we add the numbers in a non–empty prefix of any sentence in the language of G, they sum upto some non–negative value.

Some example sentences of the language of the grammar are given below (each line is a separate sentence, empty sequence is not a member of the language):

```
+1 -1
+1 -1 +1 -1
+1 -1 +1 -1 +1 -1
+1 +1 -1 -1
+1 +1 +1 -1 -1 -1
+1 +1 -1 +1 -1 -1 +1 +1 -1 -1
```

Let us use the token `P` to denote `+1` and let us use the token `M` to denote -1.

Using **only 1 nonterminal symbol** (and the tokens `P` and `M`), give an **ambiguous** context free grammar G **with at most 3 productions**, which generates this language. **Show that your grammar is ambiguous** by using a sentence with **at most 6 tokens**.

Question 3) [**5 points**] Below is a partially shown bison file. Only 8 of the productions are shown (there are more but not shown), and these are related to the parsing of the function declaration part.

==========

```
...
%union {
   int   size;
   char* lexeme;
}
%token <lexeme> tIDENT
...... <<<--- attribute declarations of nonterminals will go here
%%
...
func_decl : type tIDENT '(' parameters ')' '{' body '}'  { /* semantic action 1 */
                 printf("Function %s needs %d bytes for local variables\n", $2, $7);
                 }
;

body      : local_vars statements    { /* semantic action 2 */ }
;

local_vars: var_decl ';' local_vars  { /* semantic action 3 */ }
          |                          { /* semantic action 4 */ }
;

var_decl  : type tIDENT              { /* semantic action 5 */ }
;

type      : int                      { /* semantic action 6 */ }
          | char                     { /* semantic action 7 */ }
          | float                    { /* semantic action 8 */ }
;
...
```

==========

The semantic action of the production of the nonterminal `func_decl` is given (`semantic action 1`). As you can see, it will print out some information message, which tells the name of the function parsed, together with the total number of bytes that will be used to store all the local variables of the function. The attribute declaration for the token `tIDENT` is also given. We see that the attribute named `lexeme` is associated with the token `tIDENT`.

Your mission, should you chose to accept it (as a matter of fact, even if you chose not to accept it) is:
- to write the attribute declarations for the nonterminals `body`, `local_vars`, `var_decl`, and `type`, and
- to give the semantic actions (2,3,..., 8) of the remaining 7 productions so that the correct message is printed out when the semantic action 1 of the production of `func_decl` is executed.

You can assume that the attribute of `tIDENT` is set by the scanner to the lexeme of the identifier seen. Assume that a character value uses 1 byte, an integer value uses 4 bytes, and a float value uses 8 bytes.