University of Toronto

CSC 488/2107 Compilers & Interpreters **Spring 2000**
Midterm [15% of final mark] **February 29, 2000**

Instructions: This midterm is open book, open notes.
Non-programmable calculators allowed. No electronic communication devices allowed.
Total marks 100 - Total time is 50 minutes.
Two pages, 5 questions, **answer all questions**.

1. [25 marks] Character string constants (string literals) in the C programming language begin and end with a " (double quote) character. A string constant may directly contain any character except `\ " or newline`. String constants can also contain any of the *escape sequences*

Special Character Escapes - `\" \' \? \\ \a \b \f \n \r \t \v`
1, 2 or 3 digit Octal Escapes - `\3 \23 \342`
1 or 2 digit Hexadecimal Escapes - `\x9 \x8f \xBD`

Where each of the escape sequences becomes a single character in the string constant.
Design a lexical analysis algorithm to process string literals in C. You can describe your algorithm any way you wish, but a finite state machine is recommended. Make sure to be clear in your algorithm about successful completion and error conditions.

2 [10 marks] The C programming language standard specifies that two adjacent character string constants separated by an arbitrary amount of whitespace should be automatically concatenated by the compiler so that they behave as one character string constant.

For example "ABCD" "EFG" is the same as "ABCDEFG"

[4 marks] Describe how this language feature could be implemented during lexical analysis.

[4 marks] Describe how this language feature could be implemented during syntax analysis.

[2 marks] Which implementation is better and why.

3. [20 marks] For the fragment of C code given below, list all of the static semantic analysis checks that a compiler would make to validate the correctness of the program.

```
for( I = 0 ; I < N ; I++ ) {
    scanf("%e ", &A[ N-I-1 ][ I ] );
    if( ! ( I % 10 ) )
        B[ J++ ] = A[ N-I-1 ][ I ] ;
}
printf("Last value for B is %e\n", B[ J - 1  ] );
```

4. [25 marks] Show the Abstract Syntax Tree that would be generated for the following program in the course project language.

```
int COUNT
int SUM
COUNT = 0
SUM = 0
loop
    if COUNT > 100 then
        exit
    else  {
          int TMP
          get  TMP
          if TMP !< 0 then
             SUM = SUM + TMP
          COUNT = COUNT + 1
    }
put "Sum is " , SUM
```

5. [20 marks] Assuming the other definitions from `astDef.h` in the course project, show the typical symbol table entries that a compiler might create for the declaration.

```
/*********************************************************/
/* Data structure used to describe expressions */
/*********************************************************/

typedef struct Expn_tag {
  ExpnType  etype;   /* type of expression */
  union expnVal_tag {
    struct expnOp_tag {      /* operator         */
     OperType eoper ;     /* type of operator  */
     ExpnP    eopleft ;   /* left operand (if any ) */
     ExpnP  eopright ;  /* right operand (if any)  */
    } eop ;
    struct expnFn_tag {      /* function call */
         char *  efname ;    /* name of function */
         ArgsP   efargs ;    /* list of arguments */
    } efn ;
    LitObj   elitVal ;    /* literal constant */
    char * eident ;     /*  identifier */
    ScopeP escope ;     /*  expresion scope */
  } evalue; /* value of the expression */

} ExpnObj;
```