Department of Computer Science
Ashoka University

Programming Language Design and Implementation (PLDI): CS-1319-1

Quiz - 2: Offline *Marks: 100*
Date: *November 05, 2022* Time: *16:30 - 18:00*

Instructions:

1. The quiz can be taken from anywhere.
2. The quiz comprises one question (totalling 100 marks) and one bonus question (for 10 marks). Each question has multiple parts with marks shown for each.
3. The quiz is Open book, Open notes, and Open Internet.
4. Any copy from peers will be dealt with zero tolerance - both to get zero in the question.
5. The quiz will be available from 16:25.
6. The quiz must be submitted within 18:10.
7. No late or email or physical submission will be entertained.
8. You may choose between two options to prepare your answers:
   (a) Write the answers on paper. Take snaps and prepare a single PDF.
   (b) Alternately, you may type your answer in an editor (Notepad / WORD etc.) and save as PDF.
   - Naturally, you may use a mixed style too (like drawing a diagram on paper and include its image in the file you are typing).
   - Whatever way you prepare the answers, all should be made into a single PDF file.
9. Remember to write Name & Roll Number on every page and put pages in right order.
10. The PDF should be named with roll number and uploaded.
11. No question or doubt will be entertained. If you have any query, make your own assumptions, state them clearly in your answer and proceed.
12. Write in clear handwriting and in an unambiguous manner. If TAs have difficulty reading / understanding your answer, they will make assumptions at their best capacity to evaluate. You would not get an opportunity for explanation or rebuttal.

1. You have to develop a scientific calculator Ganak for integer arithmetic by writing Flex & Bison specifications for the same. The calculator uses integer literals and a number of the binary and unary operators as listed in the grammar below.

   The arity, precedence, associativity, and behaviour of the computations of the operators are presented through example expressions and accompanying statements.

   Ganak reads an expression involving integer literals and the operators and displays the value of the expression on the console. The expression is terminated by a $.

Sr#	Rule	Operator Semantics		
		Name	**Associativity**	**Precedence**
1:	$S \rightarrow E\ \$$		*Displays the value of expression E when $ is in the input*	
2:	$E \rightarrow E + E$	Addition[a]	`2+3+5=(2+3)+5=10`	`2+3*5=2+(3*5)=17`
3:	$E \rightarrow E - E$	Subtraction[a]	`2-3-5=(2-3)-5=-6`	`2-3*5=2-(3*5)=-13`
		`Prec(+) = Prec(-)`[b]		
4:	$E \rightarrow E * E$	Multiplication[a]	`2*3*5=(2*3)*5=30`	`2*3^2=2*(3^2)=18`

5:	$E \rightarrow E / E$	Division[a]	`6/2/3=(6/2)/3=1`	`9/3^2=9/(3^2)=1`
				`7/3=2`
				`7/(-3)=(-7)/3=-2`
				`7/0=error`

		`Prec(*) = Prec(/)`		
6:	$E \rightarrow E\ \hat{}\ E$	Exponentiation[c]	`2^3^2=2^(3^2)=2^8=512`	`-2^3=(-2)^3=-8` `2^-3=2^(-3)=1/8=0`
7:	$E \rightarrow - E$	Negation[a]	`--5=-(-5)=5`	`-2!=-(2!)=-2`
8:	$E \rightarrow + E$	Identity[a]	`++5=+(+5)=5`	`+2!=+(2!)=2`
9:	$E \rightarrow$ `sgn` E	Sign Flipper[d]	`sgn -5=5.  sgn 5=-5` `sgn sgn -5=sgn(sgn(-(5)))`	`sgn 3!=sgn (3!)=-6`
		`Prec(unary -) = Prec(unary +) = Prec(sgn)`		

10:	$E \rightarrow E$!	Factorial[e]	`3!!=(3!)!`	`3!=3*2*1=6`
				`(-3)!=-3*-2*-1=`
				`(-1)^3*3!=-6`
				`(-2)!=(-1)^2*2!=2`

11:	$E \rightarrow (E)$	Parenthesis	*Supersede precedence & associativity*	
12:	$E \rightarrow$ `num`		*Unsigned integer literal*	

[a]: *Operators* `+, -, *, /, unary +, and unary -` *follow the behaviour of their C counterpart.*
[b]: `Prec` *denotes the precedence of an operator.*
[c]: *Exponentiation is implemented by repeated multiplication by a* `power(a, b) = a^b` *function*
[d]: `sgn` *is a multi-character operator (like* `sizeof` *in C). A blank is needed after* `sgn`
[e]: *Do not read* `3!=3*2*1` *as* `3` *not equal to* `3*2*1`*. Read as* `3!`*, that is,* `3` *factorial, equals* `3*2*1`*. Factorial is implemented by a* `factorial(n)` *function*

(a) Fully parenthesize and evaluate the following expressions according to the rules of Ganak.
[2 + 2 + 3 + 3 = 10]

i. `18 - 2 * 3!  - 5`
ii. `5 + + 2 - - 3`
iii. `15 / 4 * 2 + sgn 37 / 5`
iv. `sgn 4!  + (sgn 4)!`

BEGIN SOLUTION

i. `((18 - (2 * (3!)))  - 5) = 18 - 12 - 5 = 1`
ii. `((5 + (+ 2)) - (- 3)) = 5 + 2 - (-3) = 7 + 3 = 10`
iii. `(((15 / 4) * 2) + ((sgn 37) / 5)) = (3 * 2) + (-37/5) = 6 - 7 = -1`
iv. `((sgn (4!))  + ((sgn 4)!))  = (sgn 24) + (-4)!  = -24 + 24 = 0`

END SOLUTION

(b) List the operators of Ganak in the order of increasing precedence showing their respective associativity. Justify the precedence order and associativity using the rules of Ganak.

[5 + 10 = 15]

BEGIN SOLUTION

Operators in order of increasing precedence (reading top to bottom):

```
+ -                          left associative
* /                          left associative
^                            right associative
- (unary) + (unary) sgn      right associative
!                            left associative
```

Justification: We use `u` to mean `unary` operators.

Operator	Order	Next Op.	Precedence	Associativity
`Prec( + )`	`=`	`Prec( - )`	Stated in the table	Left: `2+3+5=(2+3)+5`
`Prec( - )`	`<`	`Prec( * )`	As `2-3*5=2-(3*5)`	Left: Same as `+`
`Prec( * )`	`=`	`Prec( / )`	Stated in the table	Left: `2*3*5=(2*3)*5`
`Prec( / )`	`<`	`Prec( ^ )`	As `9/3^2=9/(3^2)`	Left: Same as `*`
`Prec( ^ )`	`<`	`Prec( u- )`	As `-2^3=(-2)^3`	Right: `2^3^2=2^(3^2)`
`Prec( u- )`	`=`	`Prec( u+ )`	Stated in the table	Right: `--5=-(-5)`
`Prec( u+ )`	`=`	`Prec( sgn )`	Stated in the table	Right: Same as `u-`
`Prec( sgn )`				Right: Same as `u-`
`Prec( u- )`	`<`	`Prec( ! )`	As `-2!=-(2!)`	
`Prec( ! )`				Left: `3!!=(3!)!`

END SOLUTION

(c) Write the Flex specifications for Ganak. **[10]**

BEGIN SOLUTION

```
%{
#include "y.tab.h" // Bison generated file of token symbols and attributes
#include <math.h>
%}

%%
"sgn"           { return SGN; }
[1-9]+[0-9]*    { yylval.intval = atoi(yytext);  return NUM; }
[ \t]           ;  /* ignore white space */
"$"             { return 0; /* end of input */ }
\n|.            return yytext[0]; /* Single character token */
%%
```

END SOLUTION

(d) Write the Bison specifications for Ganak. Assume that the following functions are available in `parser.c` source file with their prototypes in `parser.h` header file. **[25]**

Prototype	Computation
`int power(int a, int b);`	Computes $a^b \equiv a$ raised to the power of b Note: $a^{-b} = \frac{1}{a^b}$
`int factorial(int a);`	Computes $a! =$ $a * (a-1) * \cdots * 1$, if $a > 0$ 1, if $a = 0$ $a * (a+1) * \cdots * -1 = (-1)^a * (-a)!$, if $a < 0$

Note: You do not need to write the codes for the above functions.

BEGIN SOLUTION

```
%{
#include <stdio.h>
#include "parser.h"
extern int yylex();
void yyerror(char *s);
%}
%union {
    int val;
}
%token <val> NUM
%token SGN

%left '+' '-'
%left '*' '/'
%right '^'
%right UMINUS UPLUS SGN
%left '!'

%type <val> E
%%
S: E $ { printf("= %d\n", $1); }
    ;
E:    E '+' E { $$ = $1 + $3; }
    | E '-' E { $$ = $1 - $3; }
    | E '*' E { $$ = $1 * $3; }
    | E '/' E
        { if ($3 == 0)
              yyerror("div by 0");
          else
              $$ = $1 / $3;
        }
    | E '^' E { $$ = power($1, $3); }
    | '-' E %prec UMINUS
        { $$ = -$2; }
    | '+' E %prec UPLUS
        { $$ = $2; }
    | SGN E { $$ = -$2; }
    | E '!' { $$ = factorial($1); }
    | '(' E ')' { $$ = $2; }
    | NUM
    ;
%%
void yyerror(char *s) {
    printf("%s\n", s);
}

int main() {
    yyparse();
}
```

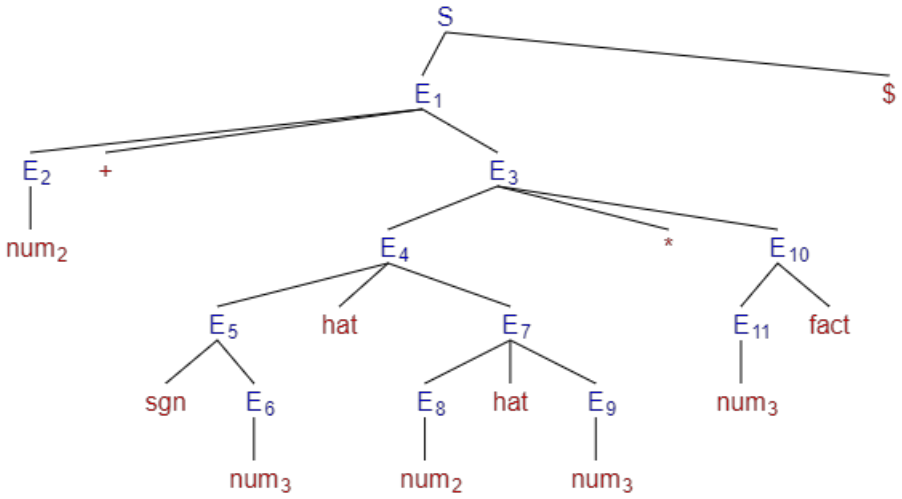
- C header: 1
- `%tokens`, `%type`, and attributes: 4
- Precedence and associativity of operators: 5
- Binary `+ - * /` Rules: 4
- Binary `^`Rule: 1
- Binary `+ - sgn /` Rules: 3
- Binary `!` Rule: 1
- Paran & `NUM` rules: 2
- `yyerror()` and `main()` functions: 2
- Overall structure: 2

END SOLUTION

(e) Consider an input `x ≡ 2 + sgn 3 ^ 2 ^ 3 * 3!$`

i. Draw the parse tree for `x` using the grammar of Ganak. [**10**]

BEGIN SOLUTION



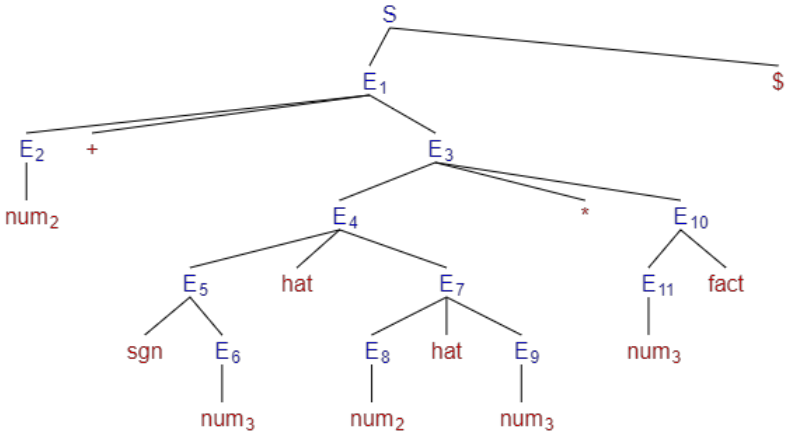
Parse Tree for `x ≡ 2 + sgn 3 ^ 2 ^ 3 * 3!$`

END SOLUTION

ii. Show the working of Ganak for `x` using the symbol and the value stacks. Show the state of the stacks for every reduction and mark the production rule used in reduction. You do not need to show the state of the stacks for shifts. [**30**]

BEGIN SOLUTION

x ≡ 2 + sgn 3 ^ 2 ^ 3 * 3!$



$	$

(0)

E	2
$	$

$E \to \texttt{num}_2$

(1)

E	3
sgn	
+	
E	2
$	$

$E \to \texttt{num}_3$

(2)

E	-3
+	
E	2
$	$

$E \to \texttt{sgn}\ E$

(3)

E	2
^	
E	-3
+	
E	2
$	$

$E \to \texttt{num}_2$

(4)

E	3
^	
E	2
^	
E	-3
+	
E	2
$	$

$E \to \texttt{num}_3$

(5)

E	2^3=8
^	
E	-3
+	
E	2
$	$

$E \to E\ \hat{}\ E$

(6)

E	$(-3)^8 = 6561$
+	
E	2
$	$

$E \to E\ \hat{}\ E$

(7)

E	3
*	
E	6561
+	
E	2
$	$

$E \to \texttt{num}_3$

(8)

E	3! = 6
*	
E	6561
+	
E	2
$	$

$E \to E\ !$

(9)

E	6561×6 = 39366
+	
E	2
$	$

$E \to E * E$

(10)

E	2+39366 = 39368
$	$

$E \to E + E$

(11)

= 39368

END SOLUTION

2. **Bonus Problem**: Consider the conditional expressions in terms of the ternary operator of C:

$$\begin{aligned}(e1)?e2:e3 &= e2,\ if\ e1 \neq 0\\ &= e3,\ otherwise\end{aligned}$$

You need to add the support for the above operator in Ganak.

(a) Write an appropriate production rule for the operator. [**2**]

BEGIN SOLUTION

$E \rightarrow (E)?E:E$

END SOLUTION

(b) Note that

```
(3 - 2 - 1)? 4 + 3: 2 + 6 * 4
```

has an ambiguity. Does it mean

```
CASE 1: (3 - 2 - 1)? 4 + 3: (2 + 6 * 4)
```

Or,

```
CASE 2: ((3 - 2 - 1)? 4 + 3: 2) + 6 * 4
```

That is, what is the precedence of the `?:` operator with respect to other operators like `+`?

i. If `CASE 1` is taken as the semantics of the above expression, what should be the precedence of the `?:` operator with respect to other operators like `+`? Justify your choice. [**2 + 1 = 3**]

BEGIN SOLUTION

The precedence of `?:` should be the least so that it is evaluated after all other operators. If it is not least then, say, operator `+`; then it will lead to CASE 2 or some other. **END SOLUTION**

ii. Since Bison does not have a simple way to define precedence of a ternary operator, how would you add and/or modify production rules to support this ternary operator in Ganak? [**4**]

BEGIN SOLUTION

$$\begin{aligned}S &\rightarrow E'\\ E' &\rightarrow (E)?E:E\\ E &\rightarrow E+E \mid E-E \mid E*E \mid E/E \mid E\hat{}E \mid -E \mid +E \mid \textbf{sgn}\ E \mid E! \mid (E) \mid \textbf{num}\\ E &\rightarrow E'\end{aligned}$$

END SOLUTION

iii. Comment on the associativity of operator `?:`. [**1**]

BEGIN SOLUTION

END SOLUTION

*The credit for a bonus problem (**10** here) is not counted in the total of **100**. However, marks scored in a bonus problem will be added to total score (capped, of course, at **100**).*