Department of Computer Science
Ashoka University

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

Quiz 3 Solutions

Marks: 100

Date: *November 20, 2023* Time: *16:40 - 18:10*

Instructions:

1. The quiz will be physical, on paper, and in the classroom.
2. Write your name and Ashoka ID on the answer-script and additional papers.
3. The quiz comprises three questions (totalling 100 marks) and one bonus question (for 10 marks). Each question has multiple parts with marks shown for each.
4. The test is entirely closed book.
5. Any copy from peers will be dealt with zero tolerance - both to get zero in the question. Consultations or chats with others will lead to zero score for the entire quiz.
6. No question or doubt will be entertained. If you have any query, make your own assumptions, state them clearly in your answer and proceed.
7. 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.

Language $\mathtt{C}_{Q3}$

Given below is the grammar for $\mathtt{C}_{Q3}$ programming language. It is a severely restricted version of the grammar of `C` programming language. It uses a subset of the token classes of `C` including the operators and literals. It has limited data types (only **int** is allowed, no array, `struct`, or `union`); only a few control constructs; no function prototype (a function is specified with its definition); and no function call. A $\mathtt{C}_{Q3}$ program comprises only one function – **main** that has no **return** statement. It supports only single line comments.

Wherever a feature is supported in $\mathtt{C}_{Q3}$, the semantics of the same follows the semantics of the feature in `C`. Note that it does not support numeric to Boolean conversion of data type – the same is achieved explicitly by $B \to E_1$ **relop** E_2.

The grammar is augmented with marker non-terminals and epsilon productions to facilitate translation as well.

Table 1: Grammar for $\mathtt{C}_{Q3}$

0:	P	$\to$	FD	// One function program. End of TAC Translate
1:	FD	$\to$	T **main ( )** BS	// Function Definition
2:	BS	$\to$	**{** $Q\ L$ **}**	// List of statements in block scope
3:	L	$\to$	$L_1\ M\ S$	// List of Statements
4:	L	$\to$	LD	// Local variable Declarations
5:	LD	$\to$	$LD_1\ VD$	// List of variable Declarations
6:	LD	$\to$	ϵ	// Empty declaration
7:	VD	$\to$	$T\ V$ **;**	// Variable declaration
8:	V	$\to$	V_1 **, id**	// List of identifiers (variables)
9:	V	$\to$	**id**	// Identifier (variable)
10:	T	$\to$	**int**	// Type (only **int** is allowed)
11:	S	$\to$	BS	// List of stmts. in a Block Scope is a single Stmt.
12:	S	$\to$	E **;**	// Any semicolon (**;**) terminated Expr. is a Stmt.
13:	S	$\to$	**if** $(B)\ M\ S_1$	// Dangling **if**
14:	S	$\to$	**if** $(B)\ M_1\ S_1\ N$ **else** $M_2\ S_2$	// Complete **if**
15:	S	$\to$	**while** $M_1\ (B)\ M_2\ S_1$	// **while** loop
16:	S	$\to$	**for** $(E_1$ **;** $M_1\ B$ **;** $M_2\ E_2\ N)\ M_3\ S_1$	// **for** loop
17:	E	$\to$	$E_1 = E_2$	// Assignment Expression
18:	E	$\to$	$E_1 + E_2$	// Addition Expression
19:	E	$\to$	$E_1 - E_2$	// Subtraction Expression
20:	E	$\to$	$E_1 * E_2$	// Multiplication Expression
21:	E	$\to$	$E_1\ /\ E_2$	// Division Expression
22:	E	$\to$	$E_1\ \%\ E_2$	// Remainder Expression
23:	E	$\to$	$-\ E_1$	// Negation Expression
24:	E	$\to$	(E_1)	// Parenthesized Expression
25:	E	$\to$	**id**	// **id** (variable) is an Expression
26:	E	$\to$	**num**	// **num** (numeric integer constant) is an Expr.
27:	B	$\to$	$B_1\ \|\|\ M\ B_2$	// Disjunction of Boolean expressions
28:	B	$\to$	$B_1\ \&\&\ M\ B_2$	// Conjunction of Boolean expressions
29:	B	$\to$	$!B_1$	// Negation of Boolean expressions
30:	B	$\to$	(B_1)	// Parenthesized Boolean expression
31:	B	$\to$	E_1 **relop** E_2	// Boolean expr. from comparison of numeric Exprs.
				// **relop** is one of $<, >, ==, !=, \leq, \geq$
32:	M	$\to$	ϵ	// Marker rule to remember TAC index
33:	N	$\to$	ϵ	// Rule to guard fall through
34:	Q	$\to$	ϵ	// Rule to create Symbol Table

1. For an Armstrong number n, the sum of the cubes of the digits of n equals n. Few small Armstrong numbers are:

$$\begin{aligned}
0 &= 0^3 \\
1 &= 1^3 \\
153 &= 1^3 + 5^3 + 3^3 \\
370 &= 3^3 + 7^3 + 0^3 \\
371 &= 3^3 + 7^3 + 1^3 \\
&\ldots
\end{aligned}$$

Consider the following program to count the number of Armstrong numbers up to a number `lim`.

```
int main() { // main block (function) -- B1
    int lim, num, cnt; // Declarations in B1

    lim = 10000; // Search limit
    cnt = 0; // Keep count

    // Check numbers from 0 to lim
    for (num = 0; num <= lim; num = num + 1) { // for block -- B2
        int copy, sum; // Declarations in B2

        copy = num; // Keep a copy of num for check later
        sum = 0;

        // Add cubes of every digit and accumulate in sum
        while (num != 0) { // while block -- B3
            int rem; // Declarations in B3

            rem = num % 10; // Find digit
            sum = sum + (rem * rem * rem); // Accumulate cubes of digits
            num = num / 10; // Reduce num
        }

        // If sum of cubes of digits of num equals to num
        // then num is Armstrong. Count up
        if (copy == sum) // if block -- B4
            cnt = cnt + 1;

        num = copy; // Restore num
    }

    // Print cnt
    //
}
```

Translate the above $\texttt{C}_{Q3}$ program to three address codes using the grammar in Table 1. For the translation, you need to use the semantic actions similar to what has been discussed in Module 7 (or as you are doing in Assignment 4) using the attributes and global variables listed in Table 2.

You do not need to write the action rules or show the actions in working. However, your translation must follow the scheme in the Module.

Note that, you may want to modify the grammar, add more attributes and global variables to assist the translation, and / or make changes to the translation scheme. If you make any change or add anything, please explain clearly with justification.

Specifically, perform the following tasks:

Table 2: Attributes and Globals for Translation of $\mathtt{C}_{Q3}$

Symbols	Attributes (Usual Meaning)
S, L, N	*nextlist*
B	*truelist*, *falselist*
E	*loc*
M	*instr*
id	*loc*
num	*val*
T, **int**	*type*, *size*

Global Variables	Meaning
nextinstr	Global counter to the array of quads
offset	Global marker for Symbol Table fill-up
type_global	Global to pass the *type* information around for inherited attributes
width_global	Global to pass the *width* information around for inherited attributes

(a) Using the grammar in Table 1, draw the following parse trees: **[2 + 5 + 7 + 6 = 20]**

i. Parse tree for `B4`.

ii. Parse tree for `B3`.

iii. Parse tree for `B2` showing trees for `B3` and `B4` just with the root.

iv. Parse tree for `B1` showing tree for `B2` just with the root.

You may want to draw the parse trees on loose sheets so that you can cross-refer to them easily. If you do that, remember to stitch all of the sheets together in right order after writing your name and Ashoka ID on each.

(b) Mark the productions on the parse tree in the order of their reduction in bottom-up parsing. Annotate the nodes of the parse trees with the attributes of symbols to show the computations during translation. **[3 + 12 = 15]**

(c) Generate the array of quad codes starting at index 100 following the order of reductions and leveraging the computations with the attributes in Table 2. **[4 + 8 + 10 + 8 = 30]**

Assume that TAC has a code `halt` which is reached when control in `main` reaches the end of the function block.

Marks breakup is shown for blocks `B4`, `B3`, `B2`, *and* `B1` *respectively.*

Ans.

```
// Block B1 TAC begins
100: t00 = 10000
101: lim = t00
102: t01 = 0
103: cnt = t01

// Block B2 TAC begins
104: t02 = 0
105: num = t02
106: if num <= lim goto 112
107: goto 136
108: t03 = 1
109: t04 = num + t03
110: num = t04
111: goto 106
```

```
112: copy = num
113: t05 = 0
114: sum = t05

// Block B3 TAC begins
115: t06 = 0
116: if num != t06 goto 118
117: goto 129
118: t07 = 10
119: t08 = num % t07
120: rem = t08
121: t09 = rem * rem
122: t10 = t09 * rem
123: t11 = sum + t10
124: sum = t11
125: t12 = 10
126: t13 = num / t12
127: num = t13
128: goto 115
// Block B3 TAC ends

// Block B4 TAC begins
129: if copy == sum goto 131
130: goto 134
131: t14 = 1
132: t15 = cnt + t14
133: cnt = t15
// Block B4 TAC ends

134: num = copy
135: goto 108
// Block B2 TAC ends

136: halt
// Block B1 TAC ends
```

(d) Write all Symbol Tables (Global ST and STs for function `main` – `B1`, `for` block – `B2` and `while` block – `B3`). For every ST show the symbol name, data type, category, size, and offset. Mark appropriate parent / child pointers to build the tree of symbol tables. [**4 + 5 + 3 + 3 = 15**]

Marks breakup is shown for STs – Global, `B1`*,* `B2`*, and* `B3` *respectively.*

Ans.

ST.glb — *Parent = Null*

Name	Type	Category	Size	Offset	Nested Block
main	void → int	func	0	0	ptr-to-main

ST.main (B1) — *Parent = ST.glb*

Name	Type	Category	Size	Offset	Nested Block
lim	int	local	4	0	null
num	int	local	4	-4	null
cnt	int	local	4	-8	null
for			0	-8	ptr-to-for-block

ST.for (B2)					*Parent = ST.main*
Name	**Type**	**Category**	**Size**	**Offset**	**Nested Block**
`copy`	int	local	4	0	null
`sum`	int	local	4	-4	null
`while`			0	-4	`ptr-to-while-block`
`if`			0	-4	`ptr-to-if-block`

ST.while (B3)					*Parent = ST.for*
Name	**Type**	**Category**	**Size**	**Offset**	**Nested Block**
`rem`	int	local	4	0	null

ST.if (B4)					*Parent = ST.for*
Name	**Type**	**Category**	**Size**	**Offset**	**Nested Block**

Solution is still considered correct if the **Nested Block** column or `for`, `while`, `if` symbols are missing from Symbol Tables.

2. We want to add the support for ternary operator expression in `C`$_{Q3}$. [**1 + 5 + 3 + 1 = 10**]

(a) Add the production rule/s (with augmentation) for the support of ternary operator expressions where only boolean expressions are allowed in condition check.
Ans.

$$E \rightarrow B \; ? \; M_1 \; E_1 \; N : M_2 \; E_2$$

(b) Write the semantic actions for the production rule/s. You may use the auxiliary functions used in Module 7 or design new functions (just state the behavior – no code is needed).
Ans.

```
$E \rightarrow B$ ? $M_1\ E_1\ N$ : $M_2\ E_2$

{
  $E.loc = gentemp();$

  $backpatch(B.truelist, M_1.instr);$
  $backpatch(B.falselist, M_2.instr);$

  // Control gets here by fall-through
  $emit(E.loc$ '=' $E_2.loc);$

  $l = makelist(nextinstr);$
  $emit($goto .... $);$

  $backpatch(N.nextlist, nextinstr);$

  $emit(E.loc$ '=' $E_1.loc);$
  $backpatch(l, nextinstr);$

}
```

(c) Using your scheme translate the following ternary expression variant of `B4`. That is:
`cnt = copy == sum ?  cnt + 1:  cnt`
Ans.

```
100: if copy == sum goto 102
101: goto 105
```

```
102: t00 = 1
103: t01 = cnt + t00
104: goto 107
105: t02 = cnt
106: goto 108
107: t02 = t01
108: cnt = t02
```

(d) Compare the above translation of `B4` with the earlier translation.

Ans.
The earlier code had 5 quads and used 2 temporaries.
This newer code has 9 quads and uses 3 temporaries.
So, the newer code produces more lines of code and uses more resources.

3. Answer the following: **[4 + 6 = 10]**

(a) What is the type of grammar given for $\texttt{C}_{Q3}$? Justify

Ans. $\texttt{C}_{Q3}$ is a Context-free Grammar as the left hand side of each production has a single non-terminal.

Rubric:

i. Correct answer, adequate reasoning : 4
ii. Correct answer, incorrect or inadequate reasoning : 1
iii. Incorrect answer : 0

(b) Consider the following grammar G,

$$\begin{aligned} S &\to abc \mid aSBc \\ cB &\to Bc \\ bB &\to bb \end{aligned}$$

Describe $L(G)$. What type of language is it?

Ans.

$$L(G) = \{a^n b^n c^n | n > 0\}.$$

The language is a Context-Sensitive Language as it is generated by a Context-Sensitive Grammar (Type - 1).

The grammar is Context-Sensitive as each production is of the form

$$\alpha A \beta \to \alpha \gamma \beta,$$

where $A \in (N \cup T)$, $\alpha, \beta, \gamma \in (N \cup T)^*$. An answer that has good reasoning but states it as recursively enumerable (Type - 0) will also be considered correct.

Rubric:

i. Correct $L(G)$, correct grammar : 6
ii. Correct $L(G)$, incorrect grammar : 4
iii. Correct grammar, incorrect $L(G)$: 2
iv. Both incorrect : 0

4. **Bonus Problem:** **[10]**

(a) Can a context-free language have a non context-free subset? Provide example or disprove.

Ans. Yes. Consider $L_1 = \{a^n b^n c^m | n > 0\}$. This language is context-free as we can define a context-free grammar $G = \langle \{S, A\}, \{a, b, c\}, S, P \rangle$ where P is given by,

$$\begin{aligned} S &\to aSbA \\ A &\to cA \mid \epsilon \end{aligned}$$

Now, consider $L_2 = \{a^n b^n c^n | n > 0\}$ from the example above. Clearly, $L_2 \subset L_1$ but is not context-free as shown earlier.

(b) Is every superset of a regular language also regular? Prove or provide counterexample.

Ans. No. Consider L_2 as above and let $L_3 = \{abc\}$. L_3 is regular and clearly, $L_3 \subset L_2$ which is not regular.

Rubric (a) + (b):

i. Correct answer with correct example: 5
ii. Correct answer with incorrect/no example: 1
iii. Incorrect answer: 0