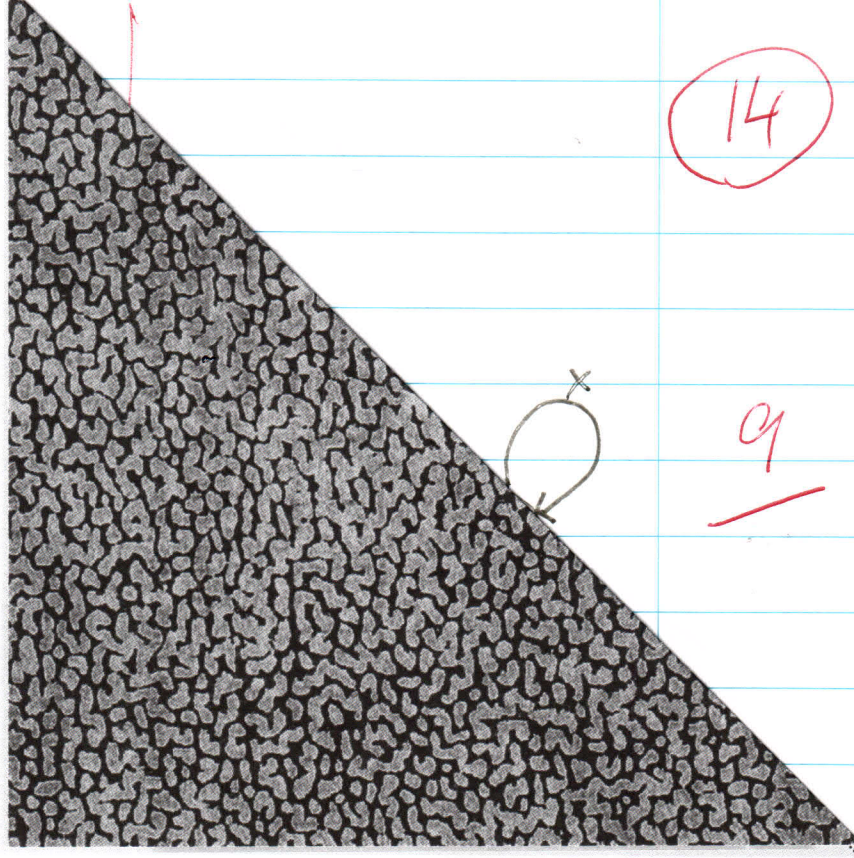
117

Coláiste na hOllscoile
Corcaigh
University College Cork



14

9

Uimhir Scrúdaithe
Examination Number

9	1	7	1	6	3	

Module Code CS4150

Paper No. _____

Mír
Section _____

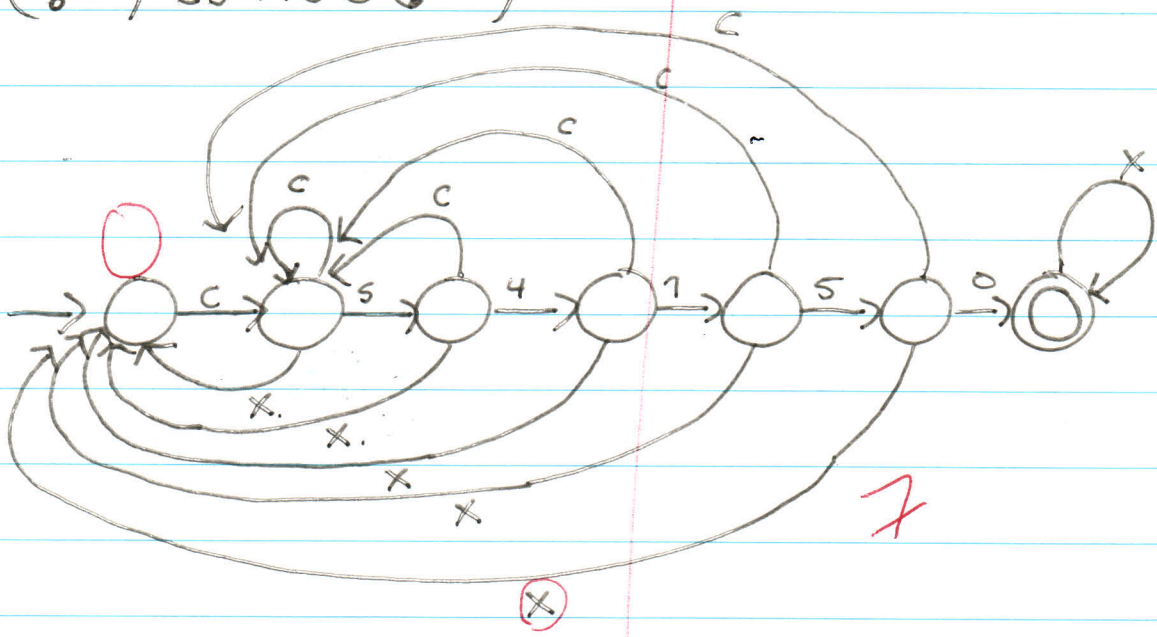
Do na Scrúdaitheóirí amháin **For Examiner's use only**	
1	14
2	24
3	30
4	10
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
Iomlán **Total**	78%

Calculator, Please state:	No. of Books submitted
Name	1
Model	

Note: If there are different sections on this paper, a separate Answer Book MUST be used for each section.

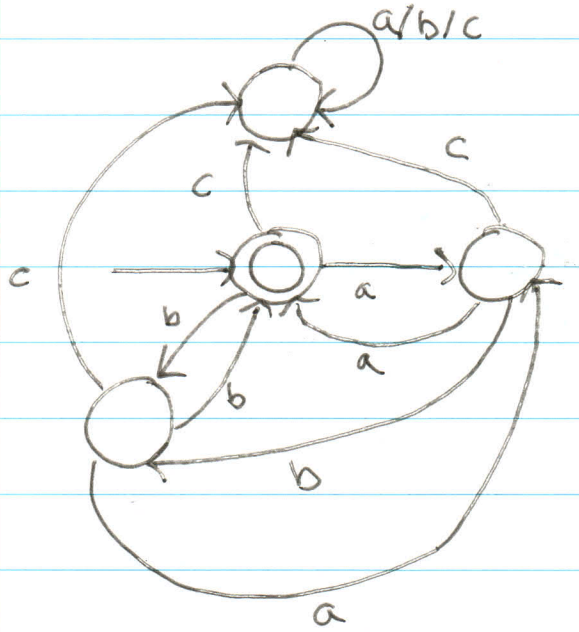
Q1

i (.*)CS4150(.*)



Where X is any symbol not already matched that is in our alphabet

Q1(ii)



abba

3/3

three #1

Q1

(iii) $(aa \mid bb)^*$

Ignores cases such as abba

2/2

Q2

(i)

5

5

$\langle G \rangle \rightarrow \langle Production \rangle \{ \backslash n \langle PRODuction \rangle \}$

(newline; zero or more)

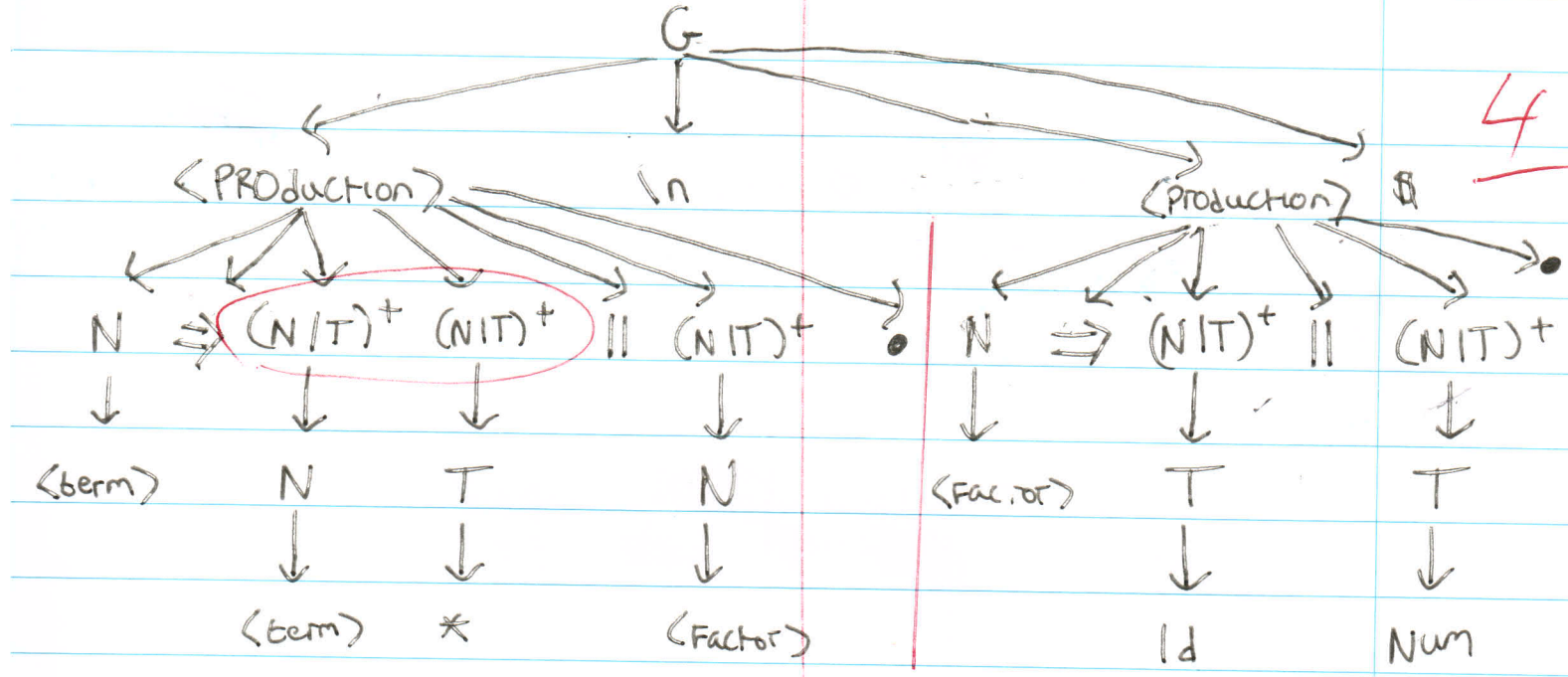
$\langle PRODuction \rangle \rightarrow N \Rightarrow (N|T)^{+} \{ || (N|T)^{+} \}$

(one or more)

24

20

(ii) Parse Tree



4

Q3

i

<Prog> → <List>

<List> → <assignment> ; <List> | ε

<assignment> → id := <expr>

<expr> → <term> { + <expr> | - <expr> } ← zero or more

<term> → <Factor> { * <term> | div <term> | mod <term> }

<Factor> → (<expr>) | id | num

Q3(ii) First set For CFG using above CFG

	First
<Prog>	First(<List>)
	⇒ {id, ε}
<List>	First(<assignment>) ∪ ε
	⇒ {id, ε}
<assignment>	⇒ {id}
<expr>	First(<term>)
	⇒ {(, id, num
<term>	First(<Factor>)
	⇒ {(, id, num}
<Factor>	⇒ {(, id, num}

Q3

iii A recursive descent parser is made of a collection of mutually recursive methods which call each other to parse the file. The methods* either recurse further to parse other non-terminals, attempt to match against expected terminals or return when they have finished matching the contents of their production

Note: methods* ⇒ A method attempts to capture the structure of a production

```
function prog():
    list()

function list():
    IF (nextToken() IS OF assignment):
        assignment()
        match(';')
        list()
    else:
        do_nothing
```

Function assignment():

```
Match(TYPE_ID)
Match(:=)
expr()
```

Note:

Match attempts to match its argument with the next token in the input symbol stream. If it succeeds it takes the token off the stream and returns
If it fails it throws an error and halts execution.
It can also take a regex as seen in Match(TYPE_ID)

NextToken returns the current value of the next token for the input stream it does not remove the value from the stream

Is of assignment, checks the token matches any tokens in First of assignment
If it does it returns true
(looks ahead 1) would need to be changed for an LL2+ parser

If Prog() returns Parsing is successfull

The expression term and factor productions could be parsed in a similar fashion

Expression() would check. IF the first input symbol matches term (errors out if not) and then check the next token to see if its a + or -, if it is ~~if~~ will call expression() if not it will return

term() will do the same checkins if the first input symbol matched factor, if it did it would check the next token to be * div or mod, if it is one of those tokens term will be called ~~again~~ otherwise it would return

Factor would check if the next token is id, num or (, if its an id or num it will return if its a (it will call expr() and then check if the next token is a)

Q4

i

PLO	TAC
sum := 0;	sum := 0
For k := 1	k := 1
to 100	t_For_1 := 100
step 1	t_For_2 := 1
do	t_For_3 := 0 //takes role of incr
begin	Start:
sum := sum + k	t_For_4 := t_For_2 * t_For_3
	IF t_For_1 <> t_For_4 go to End For
	t_val_1 := sum + k
	sum := t_val_1
	t_For_4 = t_For_3 + 1
	t_For_3 = t_For_4
	go to start
end	END FOR

10

6

Q4 ii The runtime stack is used to handle Method calls and recursion. FRame Pointers are pushed onto the stack when a method is called containing the current program counter and other information local to that Method Local Variable are then placed after this FP, When the call is about to return The FP is poped off, the PC in it is

4