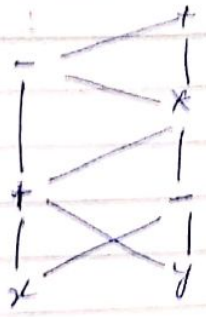
Page :
Date : / /

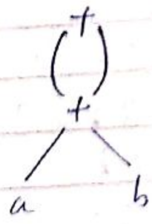
Nikeet Keshari
20174017
Compiler Assgn - 3

Q.1

(i)

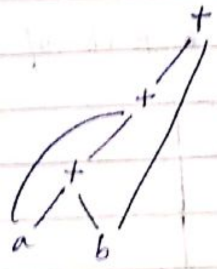


(ii)



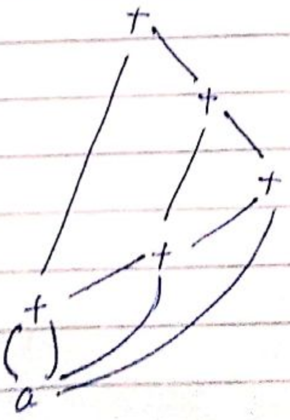
1	id	a	
2	id	b	
3	+	1	2
4	+	3	3

(iii)



1	id.	a	
2	id	b	
3	+	1	2
4	+	3	1
5	+	4	2

(iv) a + a + (a + a + a + (a + a + a + a))



1	id	a	
2	+	1	1
3	+	2	1
4	+	3	1
5	+	3	4
6	+	2	5

Q.3 (a) x = y op z and x = op y
x, y, z are the operands
"op" represents operator

It assigns the result obtained after solving right side expression of the assignment operator to the left side operand.

(b) goto X

here X is the tag or label of the target statement on executing the statement

→ control is sent directly to the location specified by label X.

→ All the statements in b/w are skipped.

(c)

```
for(i=1; i<=10; i++)        |   i = 1
{                           |   L : t1 = x * 5
   a[i] = x * 5;            |       t2 = 4 a
}                           |       t3 = sizeof(int)
                            |       t4 = t3 * i ; t5 = t2 + t4
                            |     * t5 = t1
                            |       i = i + 1
                            |     if i < 10 goto L
```

(d)

```
if ( x<100 || x > 200 && x != y)
      x = 0;
```

three address instructions:

```
     if x < 100 goto L2
     if False x > 200 goto L1
     if False x != y goto L1
L2 :  x = 0
L1 :
```

(e) if A < B then 1 else 0

Page :
Date : / /

Soln: 3 address code for given expression:

```
(1) if (A<B) goto (4)
(2)   T1 = 0
(3) goto (5)
(4)   T1 = 1
(5)
```

if A<B and C<D then t=1 else t=0

3 Address code:

```
(1) if (A<B) goto (3)
(2)  goto (4)
(3) if (C<D) goto (6)
(4)  t=0
(5)  goto (7)
(6)  t=1
(7)
```

(g)

f (4)

~~while loop~~

```
a=3;
b=4;
i=0;
while (i<n) {
    a=b+1;
    a=a*a;
    i++;
}
c=a;
```

3 Addr code:

```
     a=3;
     b=4;
     i=0;
L1:  var1 = i<n;
     if (var1) goto L2;
     goto L3;

L2:  var2 = b+1;
     a = var2;
     var3 = a*a;
     a = var3;
     i++;
     goto L1;
L3:  c = a;
```

g (2)

Page :
Date : / /

do-while loop :-

```
a = 3;
b = 4;
i = 0;
do {
    a = b + 1;
    a = a * a;
    i++;
} while (i<n);
c = a;
```

3 Addr code :-

```
    a = 3;
    b = 4;
    i = 0;
L1: var 2 = b + 1;
    a = var 2;
    var3 = a*a;
    a = var 3;
    i++;

    var 1 = (i < n);
    if(var 1) goto L1;
    goto L2;
L2: c = a;
```

Q

```
switch (ch)
{
    case 1 : c = a + b;
             break;

    case 2 : c = a-b;
             break;
}
```

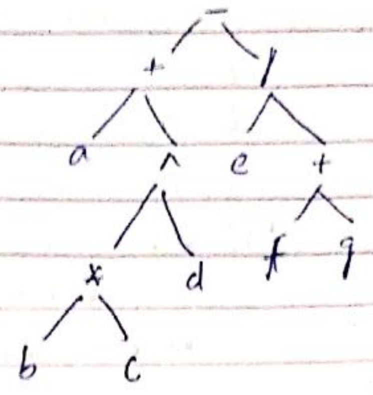
```
if Ch = 1 goto L1 ;
if ch = 2  goto L2 ;
L1 : T1 = a + b ;
     c = T1 ;
     goto Last;

L2 : T2 = a - b ;
     c = T2 ;
     goto Last ;

last
```

Q 2 Postfix notation!

abc+d^+efg+/-

Scanned with CamScanner

Syntax Tree:



Q.4 (a) Symbol table is an important data structure created & maintained by compilers in order to store info. about the occurances of various entities such as variable names, function names, objects, classes, etc. It can be used by both the analysis & synthesis parts of a compiler. May serve following purposes depending on language:-

(i) To store names of all entities in a structural form in one place
(ii) To verify if a variable has been declared.
(iii) scope resolution

operations / procedure to store names in symbol Table:-
→ insert ( )
used in analysis phase when tokens are identified & names are stored in the table.
Inserts info like the unique name ~~according~~ occuring in the source code.

(b) Data structures used:-
→ Linear (sorted / unsorted list)
→ Hash Table (most used)
→ Binary Search Tree
→ List of hash Tables
→ Hash Table of Lists

Niket Keshari
20174023