

Name: Bhavesh Kewalramani
Roll No.: A-25

Practical No. 6

Theory

Three address code is a type of intermediate code which is easy to generate and can be easily converted to machine code. It makes use of at most three addresses and one operator to represent an expression and the value computed at each instruction is stored in temporary variable generated by compiler. The compiler decides the order of operation given by three address code.

General representation –
$$a = b \text{ op } c$$

Where a, b or c represents operands like names, constants or compiler generated temporaries and op represents the operator

Example-1: Convert the expression $a * - (b + c)$ into three address code.

$$\begin{aligned} t_1 &= b + c \\ t_2 &= \text{uminus } t_1 \\ t_3 &= a * t_2 \end{aligned}$$

Example-2: Write three address code for following code

```
for(i = 1; i<=10; i++)
```

```
{  
    a[i] = x * 5;  
}
```

```

    i = 1
L : t1 = x * 5
    t2 = &a
    t3 = sizeof(int)
    t4 = t3 * i
    t5 = t2 + t4
    *t5 = t1
    i = i + 1
    if i <= 10 goto L

```

Implementation of Three Address Code –

There are 3 representations of three address code namely

1. Quadruple
2. Triples
3. Indirect Triples

1. Quadruple –

It is structure with consist of 4 fields namely op, arg1, arg2 and result. op denotes the operator and arg1 and arg2 denotes the two operands and result is used to store the result of the expression.

Advantage –

- Easy to rearrange code for global optimization.
- One can quickly access value of temporary variables using symbol table.

Disadvantage –

- Contain lot of temporaries.
- Temporary variable creation increases time and space complexity.

Example – Consider expression $a = b * -c + b * -c$.

The three address code is:

```

t1 = uminus c
t2 = b * t1
t3 = uminus c
t4 = b * t3
t5 = t2 + t4
a = t5

```

#	Op	Arg1	Arg2	Result
(0)	uminus	c		t1
(1)	*	t1	b	t2
(2)	uminus	c		t3
(3)	*	t3	b	t4
(4)	+	t2	t4	t5
(5)	=	t5		a

Quadruple representation

2. Triples –

This representation doesn't make use of extra temporary variable to represent a single operation instead when a reference to another triple's value is needed, a pointer to that triple is used. So, it consist of only three fields namely op, arg1 and arg2.

Disadvantage –

- Temporaries are implicit and difficult to rearrange code.
- It is difficult to optimize because optimization involves moving intermediate code. When a triple is moved, any other triple referring to it must be updated also. With help of pointer one can directly access symbol table entry.

Example – Consider expression $a = b * -c + b * -c$

#	Op	Arg1	Arg2
(0)	uminus	c	
(1)	*	(0)	b
(2)	uminus	c	
(3)	*	(2)	b
(4)	+	(1)	(3)
(5)	=	a	(4)

Triples representation

3. Indirect Triples –

This representation makes use of pointer to the listing of all references to computations which is made separately and stored. Its similar in utility as compared to quadruple representation but requires less space than it.

Temporaries are implicit and easier to rearrange code.

Example – Consider expression $a = b * -c + b * -c$

List of pointers to table

#	Op	Arg1	Arg2
(14)	uminus	c	
(15)	*	(14)	b
(16)	uminus	c	
(17)	*	(16)	b
(18)	+	(15)	(17)
(19)	=	a	(18)

#	Statement
(0)	(14)
(1)	(15)
(2)	(16)
(3)	(17)
(4)	(18)
(5)	(19)

Indirect Triples representation

Question – Write quadruple, triples and indirect triples for following expression
 $(x + y) * (y + z) + (x + y + z)$

Explanation – The three address code is:

$t1 = x + y$

$t2 = y + z$

$t3 = t1 * t2$

$t4 = t1 + z$

$t5 = t3 + t4$

#	Op	Arg1	Arg2	Result
(1)	+	x	y	t1
(2)	+	y	z	t2
(3)	*	t1	t2	t3
(4)	+	t1	z	t4
(5)	+	t3	t4	t5

Quadruple representation

#	Op	Arg1	Arg2
(1)	+	x	y
(2)	+	y	z
(3)	*	(1)	(2)
(4)	+	(1)	z
(5)	+	(3)	(4)

Triples representation

List of pointers to table

#	Op	Arg1	Arg2
(14)	+	x	y
(15)	+	y	z
(16)	*	(14)	(15)
(17)	+	(14)	z
(18)	+	(16)	(17)

#	Statement
(1)	(14)
(2)	(15)
(3)	(16)
(4)	(17)
(5)	(18)

Indirect Triples representation

Practicals

Aim:

Write a program to generate three address code for the given language construct using SDTS.

- (a) if-then-else
- (b) repeat-until with if then else

(a)

Program:

```
import os
```

```
global ft
```

```

with open("if.txt", "r") as input:
    with open("temp.txt", "w") as output:
        for line in input:
            if not (line.strip("\n").startswith('{') or line.strip("\n").startswith('}')):
                output.write(line)

```

```

rules=[]
with open("temp.txt", "r") as temp:
    for line in temp:
        line=line.replace("\n", "")
        rules.append([line])

```

```

p=1
ft=rules[0]

```

```

for i in range(len(rules)):
    if "=" in rules[i][0]:
        rul=rules[i][0].split("=")
        rules[i]=rul
    if "(" in rules[i][0]:
        rul=rules[i][0].split("(")
        rul[1]=rul[1].replace(")", "")
        rules[i]=rul

```

```

if "|" in rules[0][1]:
    temp=rules[0]
    rul=rules[0][1].split("|")
    for j in range(len(rul)):
        rules.insert(j,["if",rul[j]])
    rules.remove(temp)

```

```

if "&&" in rules[0][1]:
    temp=rules[0]
    rul=rules[0][1].split("&&")
    for j in range(len(rul)):
        rules.insert(j,["if",rul[j]])
    rules.remove(temp)

```

```

print(rules)
y=0
if "|" in ft[0]:
    j=0
    f=1
    res={}
    m=j
    for i in range(len(rules)):

```

```

ans=""
if "if" in rules[i][0]:
    m=j
    ans="if " + rules[i][1] + " goto "+str((len(rules)%4)+1)
    res[j]=ans
    j+=1
    ans="goto " + str(y+2)
    res[j]=ans
    j+=1
    y+=len(rules)
elif "else" in rules[i]:
    m=j
    ans="goto " + str(m+len(rules[i:])+2)
    res[j]=ans
    j+=1
elif len(rules[i])==2:
    rules[i][1]=rules[i][1].strip()
    ans="t"+str(f)+" = "+rules[i][1]
    res[j]=ans
    j+=1
    ans=rules[i][0]+" = "+"t"+str(f)
    res[j]=ans
    j+=1
    f+=1

res[j]="END"
for k,v in res.items():
    print(k, " ",v)

elif "&&" in ft[0]:
    j=0
    f=1
    res={}
    m=j
    for i in range(len(rules)):
        ans=""
        if "if" in rules[i][0]:
            m=j
            ans="if " + rules[i][1] + " goto "+str(m+2)
            res[j]=ans
            j+=1
            ans="goto " + str(len(rules))
            res[j]=ans
            j+=1
        elif "else" in rules[i]:
            m=j

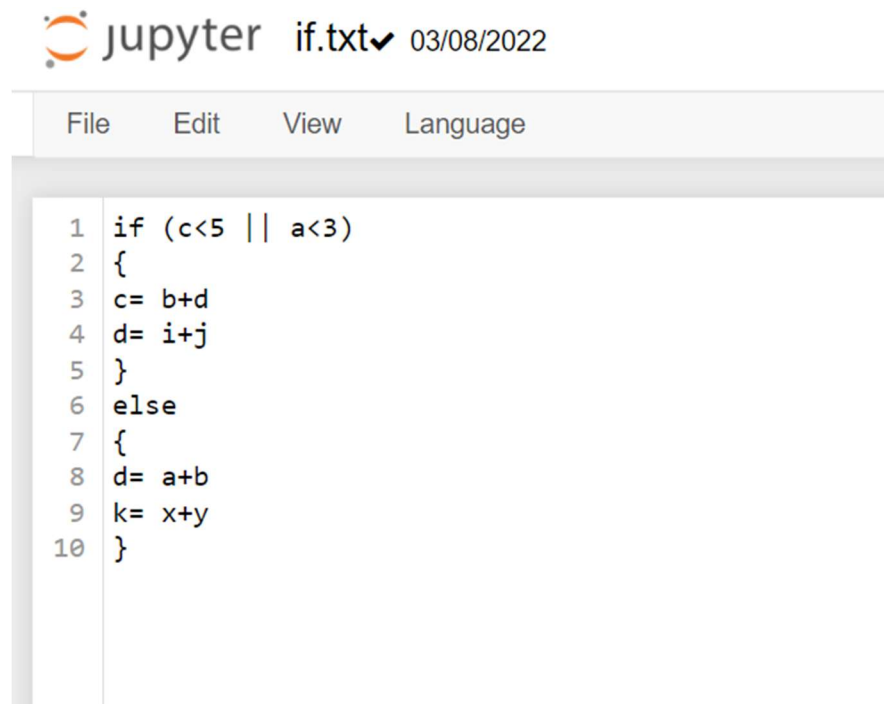
```

```

        ans="goto " + str(m+len(rules[i:])+2)
        res[j]=ans
        j+=1
    elif len(rules[i])==2:
        rules[i][1]=rules[i][1].strip()
        ans="t"+str(f)+" = "+rules[i][1]
        res[j]=ans
        j+=1
        ans=rules[i][0]+" = "+"t"+str(f)
        res[j]=ans
        j+=1
        f+=1

res[j]="END"
for k,v in res.items():
    print(k, " ",v)

```

Input:


The image shows a Jupyter Notebook interface. At the top, there is a header bar with the Jupyter logo, the text 'jupyter if.txt', a checkmark, and the date '03/08/2022'. Below the header is a menu bar with 'File', 'Edit', 'View', and 'Language'. The main area of the notebook displays a Python code snippet in a monospaced font, with line numbers 1 through 10 on the left margin. The code is an if-else statement that checks if 'c' is less than 5 or 'a' is less than 3. If true, it assigns 'c' to 'b+d' and 'd' to 'i+j'. If false, it assigns 'd' to 'a+b' and 'k' to 'x+y'.

```

1  if (c<5 || a<3)
2  {
3  c= b+d
4  d= i+j
5  }
6  else
7  {
8  d= a+b
9  k= x+y
10 }

```

Output:


```

print(r, v)

[['if', 'c<5 '], ['if', ' a<3'], ['c', ' b+d'], ['d', ' i+j'], ['else'], ['d', ' a+b'], ['k', ' x+y']]
0  if c<5 goto 4
1  goto 2
2  if a<3 goto 4
3  goto 9
4  t1 = b+d
5  c = t1
6  t2 = i+j
7  d = t2
8  goto 13
9  t3 = a+b
10 d = t3
11 t4 = x+y
12 k = t4
13 END

```

(b)

Program

```
import os
```

```
global ft
```

```
with open("repeat.txt", "r") as input:
```

```
    with open("templ.txt", "w") as output:
```

```
        for line in input:
```

```
            if not (line.strip("\n").startswith('{') or line.strip("\n").startswith('}')):
```

```
                output.write(line)
```

```
rules=[]
```

```
with open("templ.txt", "r") as temp:
```

```
    for line in temp:
```

```
        line=line.replace("\n", "")
```

```
        rules.append([line])
```

```
p=1
```

```
ft1=[rules[0], rules[-1]]
```

```
rules.remove(rules[0])
```

```
rules.remove(rules[-1])
```

```
ft=rules[0]
```

```

varr = ft1[1]

for i in range(len(ft1)):
    if "(" in ft1[i][0]:
        f=ft1[i][0].split("(")
        f[1]=f[1].replace(")", "")
        ft1[i]=f

    if "&&" in ft1[1][1]:
        temp=ft1[1]
        rul=ft1[1][1].split("&&")
        for j in range(len(rul)):
            ft1.insert(j+1,["until",rul[j]])
        ft1.remove(temp)

    if "|" in ft1[1][1]:
        temp=ft1[1]
        rul=ft1[1][1].split("|")
        for j in range(len(rul)):
            ft1.insert(j+1,["until",rul[j]])
        ft1.remove(temp)

print(rules)

for i in range(len(rules)):
    if "(" in rules[i][0]:
        rul=rules[i][0].split("(")
        rul[1]=rul[1].replace(")", "")
        rules[i]=rul

    if "=" in rules[i][0]:
        rul=rules[i][0].split("=")
        rules[i]=rul

    if "|" in rules[0][1]:
        temp=rules[0]
        rul=rules[0][1].split("|")
        for j in range(len(rul)):
            rules.insert(j,["if",rul[j]])
        rules.remove(temp)

    if "&&" in rules[0][1]:

```

```

temp=rules[0]
rul=rules[0][1].split("&&")
for j in range(len(rul)):
    rules.insert(j,["if",rul[j]])
rules.remove(temp)

print(rules)

y=0
a1=0
if "|" in ft[0]:
    j=0
    f=1
    res={}
    m=j
    for i in range(len(rules)):
        ans=""
        if "if" in rules[i][0]:
            m=j
            ans="if " + rules[i][1] + " goto "+str((j%len(rules))+2)
            res[j]=ans
            j+=1
            ans="goto " + str((y+2))
            res[j]=ans
            j+=1
            y+=len(rules)
        elif "else" in rules[i]:
            m=j
            ans="goto " + str(m+len(rules[i:])+2)
            res[j]=ans
            j+=1
        elif len(rules[i])==2:
            rules[i][1]=rules[i][1].strip()
            ans="t"+str(f)+"="+rules[i][1]
            res[j]=ans
            j+=1
            ans=rules[i][0]+"="+str(f)
            res[j]=ans
            j+=1
            f+=1
    if "|" in varr[0]:
        ans="if "+str(ft1[1][1])+" goto "+"_____"
        res[j]=ans
        j+=1
        ans="goto " + str(j+1)
        res[j]=ans

```

```

j+=1
ans="if "+str(ft1[2][1])+" goto "+"_____"
res[j]=ans
j+=1
ans="goto " + str(a1)
res[j]=ans
j+=1

if "&&" in varr[0]:
    ans="if "+str(ft1[1][1])+" goto "+str(j+len(ft1)-1)
    res[j]=ans
    j+=1
    ans="goto " + str(a1)
    res[j]=ans
    j+=1
    ans="if "+str(ft1[2][1])+" goto "+"_____"
    res[j]=ans
    j+=1
    ans="goto " + str(a1)
    res[j]=ans
    j+=1

res[j]="END"
for k,v in res.items():
    print(k," ",v)

elif "&&" in ft[0]:
    j=0
    f=1
    res={}
    m=j
    for i in range(len(rules)):
        ans=""
        if "if" in rules[i][0]:
            m=j
            ans="if " + rules[i][1] + " goto "+str(m+2)
            res[j]=ans
            j+=1
            ans="goto " + str(len(rules))
            res[j]=ans
            j+=1
        elif "else" in rules[i]:
            m=j
            ans="goto " + str(m+len(rules[i:])+2)
            res[j]=ans
            j+=1

```

```

elif len(rules[i])==2:
    rules[i][1]=rules[i][1].strip()
    ans="t"+str(f)+" = "+rules[i][1]
    res[j]=ans
    j+=1
    ans=rules[i][0]+" = "+str(f)
    res[j]=ans
    j+=1
    f+=1

```

```

if "|" in varr[0]:
    ans="if "+str(ft1[1][1])+" goto "+"_____"
    res[j]=ans
    j+=1
    ans="goto " + str(j+1)
    res[j]=ans
    j+=1
    ans="if "+str(ft1[2][1])+" goto "+"_____"
    res[j]=ans
    j+=1
    ans="goto " + str(a1)
    res[j]=ans
    j+=1

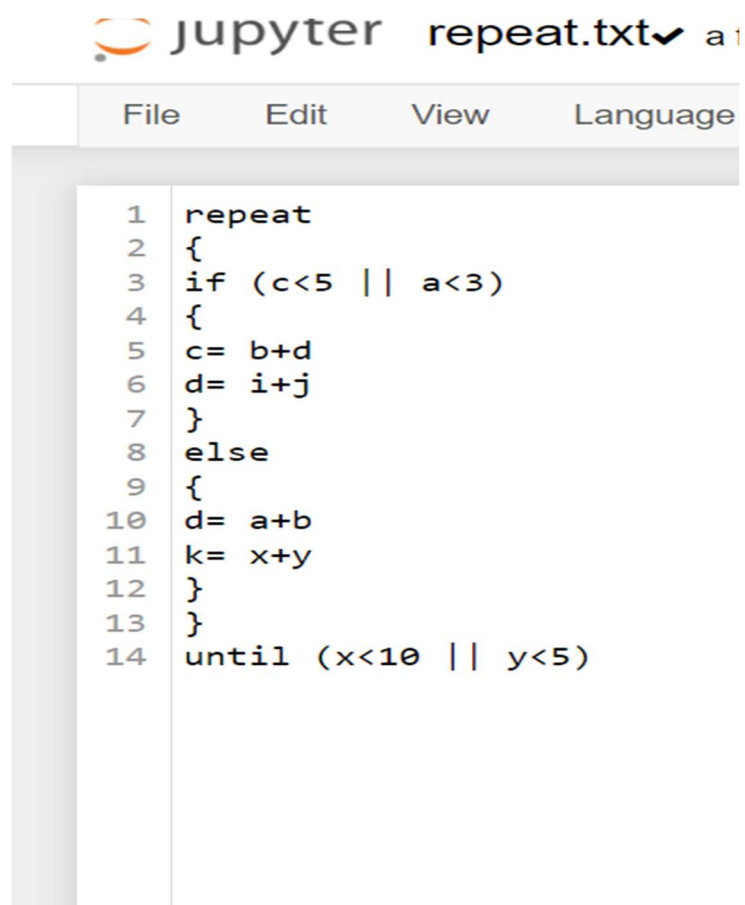
```

```

if "&&" in varr[0]:
    ans="if "+str(ft1[1][1])+" goto "+str(j+len(ft1)-1)
    res[j]=ans
    j+=1
    ans="goto " + str(a1)
    res[j]=ans
    j+=1
    ans="if "+str(ft1[2][1])+" goto "+"_____"
    res[j]=ans
    j+=1
    ans="goto " + str(a1)
    res[j]=ans
    j+=1
res[j]="END"
for k,v in res.items():
    print(k," ",v)

```

Input:



The image shows a Jupyter Notebook interface. At the top, there is a header bar with the Jupyter logo and the text 'repeat.txt' followed by a dropdown arrow. Below the header bar is a menu bar with the options 'File', 'Edit', 'View', and 'Language'. The main area of the notebook is a code editor displaying a code snippet. The code is as follows:

```
1 repeat
2 {
3   if (c<5 || a<3)
4   {
5     c= b+d
6     d= i+j
7   }
8   else
9   {
10    d= a+b
11    k= x+y
12  }
13 }
14 until (x<10 || y<5)
```

Output:

```

[['if (c<5 || a<3)'], ['c= b+d'], ['d= i+j'], ['else'], ['d= a+b'], ['k= x+y']]
[['if', 'c<5 '], ['if', ' a<3'], ['c', ' b+d'], ['d', ' i+j'], ['else'], ['d', ' a+b'], ['k', ' x+y']]
0  if c<5 goto 2
1  goto 2
2  if a<3 goto 4
3  goto 9
4  t1 = b+d
5  c = t1
6  t2 = i+j
7  d = t2
8  goto 13
9  t3 = a+b
10 d = t3
11 t4 = x+y
12 k = t4
13 if x<10 goto ____
14 goto 15
15 if y<5 goto ____
16 goto 0
17 END

```

]: