BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
(Empowered Autonomous Institute Affiliated to University of Mumbai)
[Knowledge is Nectar]

Department of Computer Engineering

Course - System Programming and Compiler Construction (SPCC)

UID	2021300101
Name	Adwait Purao
Class and Batch	TE Computer Engineering - Batch B
Date	05/04/2024
Lab #	6
Aim	Write a program to find Basic Blocks and generate flow graph for given three address code
Objective	1. Optimize code by identifying basic blocks and applying transformations. 2. Visualize control flow using flow graphs to enhance optimization strategies
Theory	**Basic Blocks:** A basic block is a sequence of consecutive statements in a program that has a single entry and exit point (Aho et al., 2006)[1]. Understanding these blocks is essential for analyzing the program's control flow. **Labels:** Labels act as markers within the code, identifying significant locations or destinations for control flow instructions such as 'GOTO' or conditional branches. They are crucial for referencing during branching and looping within the code (Appel, 2002)[2]. **Control Flow:** By identifying basic blocks and labels, we can thoroughly examine the program's control flow. Labels typically denote the start of each basic block, while control flow instructions guide the transitions between these blocks (Muchnick, 1997)[3]. **Complexity:** Complexity is introduced through conditional branches, loops, and additional branching points, which increase the number of basic blocks and complicate the control flow (Aho et al., 2006)[1]. **Analysis:** The analysis involves understanding the interconnections between basic blocks via control flow instructions, and how labels facilitate these connections. It also includes identifying conditions that alter the control flow within the program. This experimentation with basic blocks and labels in code provides deep insights into the structure and control flow of programs, crucial for program behavior analysis, debugging, and optimization (Appel, 2002)[2].

Basic Blocks in Compiler Design:

- A basic block is essentially a straight-line code sequence with no branches, except at the entry and exit points (Aho et al., 2006)[1].
- The initial task is to divide a sequence of three-address codes into basic blocks (Appel, 2002)[2].
- A new basic block starts with the first instruction and continues until a jump or label is encountered (Muchnick, 1997)[3].
- Without a jump, control proceeds directly from one instruction to the next (Aho et al., 2006)[1].

Algorithm for Partitioning:

Input: A sequence of three-address instructions.
Process: Determine 'leaders'—instructions from which basic blocks begin.
Leaders are:

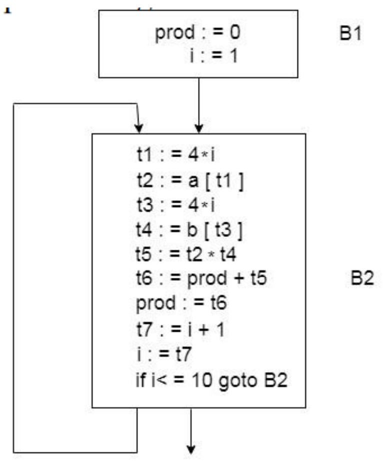
1. The first three-address instruction (Aho et al., 2006)[1].
2. Instructions that are targets of unconditional or conditional jumps (Appel, 2002)[2].
3. Instructions immediately following a jump (Muchnick, 1997)[3].
4. Each leader defines a basic block, which includes all subsequent instructions up to (but not including) the next leader (Aho et al., 2006)[1].

Example:
Consider this sequence forming a basic block:

```
t1 := aa;
t2 := ab;
t3 := 2t2;
t4 := t1+t3;
t5 := bb;
t6 := t4+t5.
```

Flow Graph:

- A control flow graph is a directed graph that maps the flow of control across various basic blocks. [4]
- It is particularly useful in optimizations like loop tuning. For instance, in a flow graph for a vector dot product:



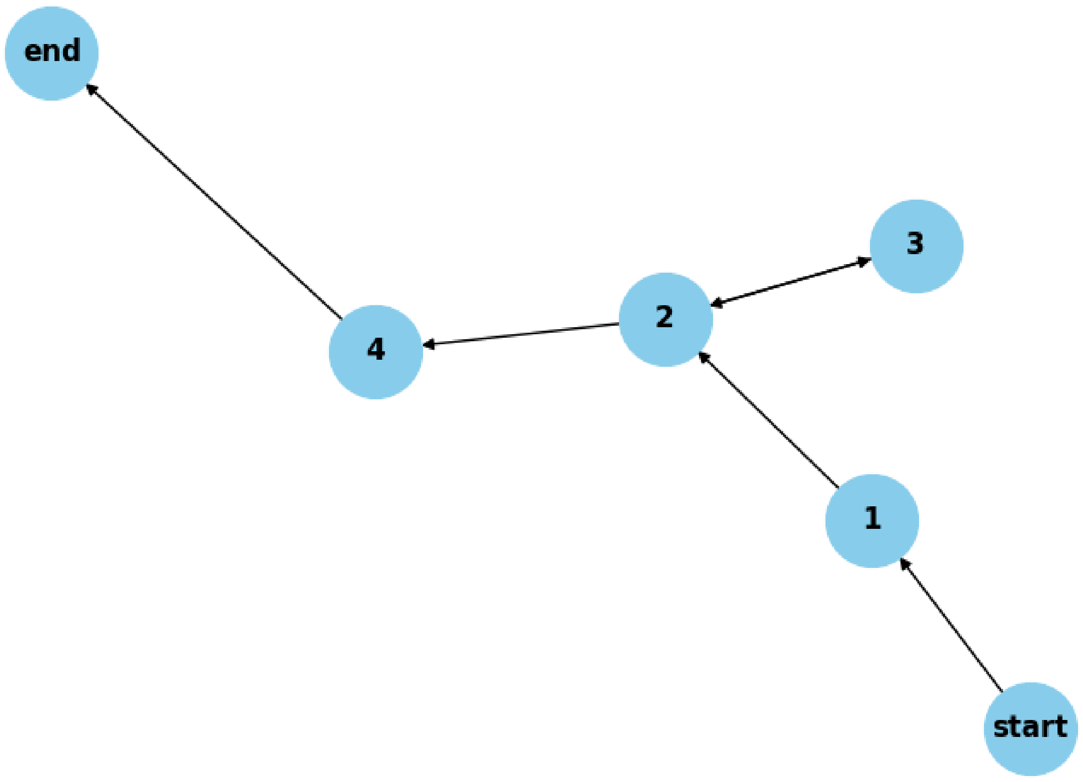
- Block B1 is the initial node.[4]
- Block B2 follows B1, creating a directional edge from B1 to B2 due to the control jump from the last statement of B1 to the first statement of B2.[4]
- B2 is a successor of B1, and B1 is a predecessor of B2.[4]

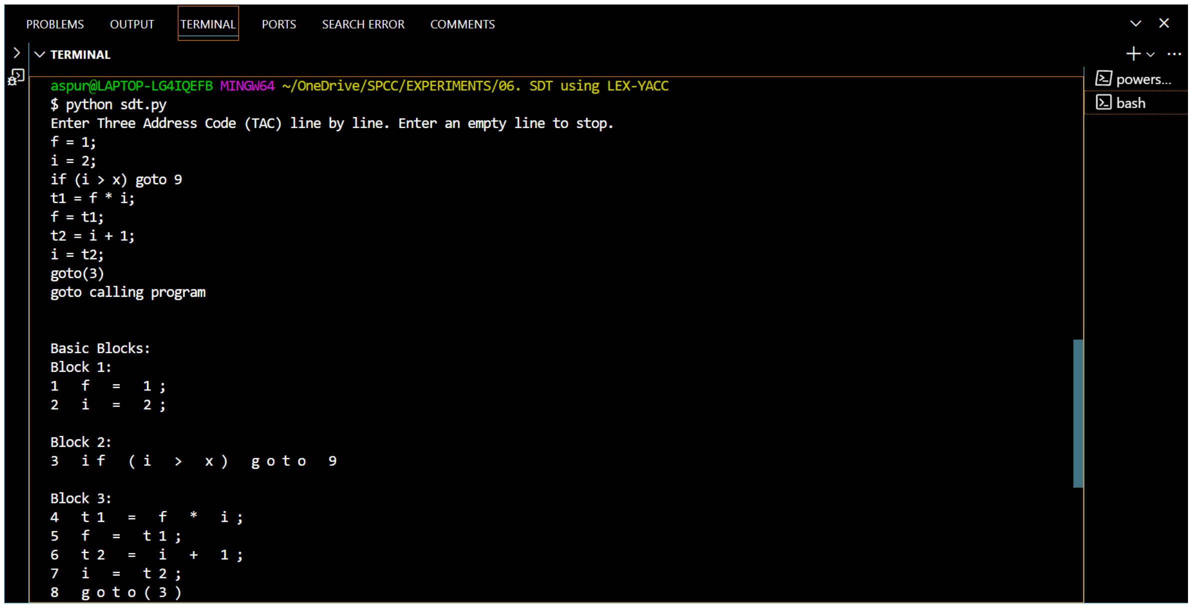
Implementation / Code

```python
import networkx as nx
import matplotlib.pyplot as plt

def generate_basic_blocks(tac):
    basic_blocks = []
    block = []

    for line in tac:
        if 'if' in line or 'goto calling program' in line or 'return'
in line:
            if block:
                basic_blocks.append(block)
            block = [line]
            basic_blocks.append(block)
            block = []
        else:
            block.append(line)

    if block:
        basic_blocks.append(block)
    return basic_blocks

def generate_control_flow_graph(basic_blocks):
  G = nx.DiGraph()
  G.add_edge('start', 1)

  for idx, block in enumerate(basic_blocks):
    last_line = block[-1]

    if 'if' in last_line:
      if 'goto' in last_line:
        true_label = last_line[-1]

      G.add_edge(idx + 1, basic_blocks.index(block) + 2)

      for next_block in basic_blocks:
        if next_block[0][0] == true_label:
          G.add_edge(idx + 1, basic_blocks.index(next_block) + 1)

    elif 'goto' in last_line:
      if 'calling' in last_line and 'program' in last_line:
        pass
      else:
        if '(' or ')' in last_line:
           last_line = last_line.replace('(', '').replace(')', '')

        next_block_label = last_line
        for next_block in basic_blocks:
          if next_block[0][0] == next_block_label[-1]:
            G.add_edge(idx + 1, basic_blocks.index(next_block) + 1)



    else:
      next_block_idx = idx + 1
      if next_block_idx < len(basic_blocks):
        G.add_edge(idx + 1, next_block_idx + 1)

  G.add_edge(idx+1,'end')
  return G

def display_control_flow_graph(G):
    pos = nx.spring_layout(G)
    pos = {node: (y, -x) for node, (x, y) in pos.items()}  # Rotate
layout
    nx.draw(G, pos, with_labels=True, node_size=1500,
node_color="skyblue", font_size=12, font_weight="bold")
    plt.title("Control Flow Graph")
    plt.show()

def main():
    tac = []
    line_number = 1
    print("Enter Three Address Code (TAC) line by line. Enter an empty
line to stop.")
    while True:
        line = input().strip()
        if not line:
            break

        tac.append(f"{line_number} {line}")
        line_number += 1


    basic_blocks = generate_basic_blocks(tac)
    control_flow_graph = generate_control_flow_graph(basic_blocks)

    print("\nBasic Blocks:")
    for i, block in enumerate(basic_blocks):
        print(f"Block {i + 1}:")
        for code in block:
```

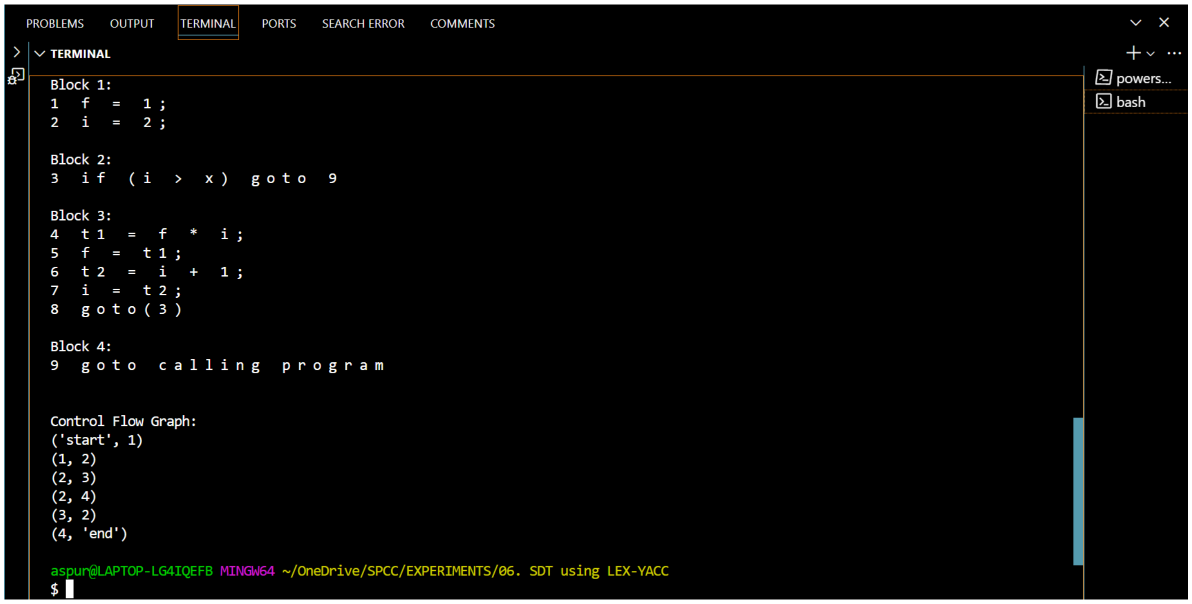
```python
            print(' '.join(code))
        print()

    print("\nControl Flow Graph:")
    for edge in control_flow_graph.edges():
        print(edge)

    display_control_flow_graph(control_flow_graph)

if __name__ == "__main__":
    main()
```

Output





```
aspur@LAPTOP-LG4IQEFB MINGW64 ~/OneDrive/SPCC/EXPERIMENTS/06. SDT using LEX-YACC
$ python sdt.py
Enter Three Address Code (TAC) line by line. Enter an empty line to stop.
f = 1;
i = 2;
if (i > x) goto 9
t1 = f * i;
f = t1;
t2 = i + 1;
i = t2;
goto(3)
goto calling program


Basic Blocks:
Block 1:
1  f  =  1 ;
2  i  =  2 ;

Block 2:
3  i f  ( i  >  x )  g o t o  9

Block 3:
4  t 1  =  f  *  i ;
5  f  =  t 1 ;
6  t 2  =  i  +  1 ;
7  i  =  t 2 ;
8  g o t o ( 3 )
```



```
Block 1:
1  f  =  1 ;
2  i  =  2 ;

Block 2:
3  i f  ( i  >  x )  g o t o  9

Block 3:
4  t 1  =  f  *  i ;
5  f  =  t 1 ;
6  t 2  =  i  +  1 ;
7  i  =  t 2 ;
8  g o t o ( 3 )

Block 4:
9  g o t o  c a l l i n g  p r o g r a m


Control Flow Graph:
('start', 1)
(1, 2)
(2, 3)
(2, 4)
(3, 2)
(4, 'end')

aspur@LAPTOP-LG4IQEFB MINGW64 ~/OneDrive/SPCC/EXPERIMENTS/06. SDT using LEX-YACC
$
```

Conclusion

Hence, we conclude that through this laboratory exercise, we have successfully devised a Python program capable of identifying Basic Blocks and constructing a control flow graph based on specified three-address code. By systematically analyzing the three-address code, we determined basic blocks using predefined criteria. Leveraging the NetworkX library, we then orchestrated a directed graph illustrating the flow of control among these basic blocks. This resultant control flow graph serves as a visual representation of the program's control dynamics, facilitating a deeper understanding of its structural intricacies and enabling analysis of its operational behavior.

References	[1] Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. (2006). Compilers: Principles, Techniques, and Tools (2nd Edition). Pearson. [2] Appel, A. W. (2002). Modern Compiler Implementation in Java (2nd Edition). Cambridge University Press. [3] Muchnick, S. S. (1997). Advanced Compiler Design and Implementation. Morgan Kaufmann. [4] GfG. (2022, March 06). Flow Graph in Code Generation. GeeksforGeeks. https://www.geeksforgeeks.org/flow-graph-in-code-generation/ [5] GfG. (2023, April 24). Compiler Design | Detection of a Loop in Three Address Code. GeeksforGeeks. https://www.geeksforgeeks.org/compiler-design-detection-of-a-loop-in-three-address-code