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)

Aim	Write a program to find Basic blocks and generate flow graph for the given three address code.
Objective	The objective of this program is to identify basic blocks and generate a flow graph for a given three-address code. The program will parse the code, identify basic blocks based on control flow statements and labels, and construct a flow graph representing the control flow between these basic blocks.
Theory	The experiment involves analyzing a given piece of code to identify its basic blocks and associated labels. Here's a theoretical breakdown of the experiment: 1. Basic Blocks: Basic blocks are consecutive sequences of statements in a program with a single entry point and a single exit point. These blocks help in understanding the control flow of the program. 2. Labels: Labels are markers within the code that indicate specific points of interest or destinations for control flow instructions like 'GOTO' or conditional branches. They provide reference points for branching and looping within the program. 3. Control Flow: By identifying basic blocks and labels, we can analyze the control flow of the program. Labels mark the beginning of each basic block, and control flow instructions dictate the flow between these blocks. 4. Complexity: Adding complexity to the code involves introducing conditional branches, loops, and additional branching points. This increases the number of basic blocks and the complexity of the control flow. 5. Analysis: The analysis of the code involves understanding how the basic blocks are connected through control flow instructions and how labels facilitate this connection. It also involves identifying the conditions under which control flow changes within the program. Overall, by experimenting with basic blocks and labels in code, we gain insights into the structure and control flow of programs, which is crucial for understanding program behavior, debugging, and optimization. **Basic Blocks in Compiler Design** Basic Block is a straight line code sequence that has no branches in and out branches except to the entry and at the end respectively. Basic Block is a set of

statements that always executes one after other, in a sequence. The first task is to partition a sequence of three-address codes into basic blocks. A new basic block is begun with the first instruction and instructions are added until a jump or a label is met. In the absence of a jump, control moves further consecutively from one instruction to another. The idea is standardized in the algorithm below:

Algorithm: Partitioning three-address code into basic blocks.

Input: A sequence of three address instructions.

Process: Instructions from intermediate code which are leaders are determined. The following are the rules used for finding a leader:

1. The first three-address instruction of the intermediate code is a leader.
2. Instructions that are targets of unconditional or conditional jump/goto statements are leaders.
3. Instructions that immediately follow unconditional or conditional jump/goto statements are considered leaders.

Each leader thus determined its basic block contains itself and all instructions up to excluding the next leader.

Example 1:

The following sequence of three-address statements forms a basic block:

t1 := a*a
t2 := a*b
t3 := 2*t2
t4 := t1+t3
t5 := b*b
t6 := t4 +t5

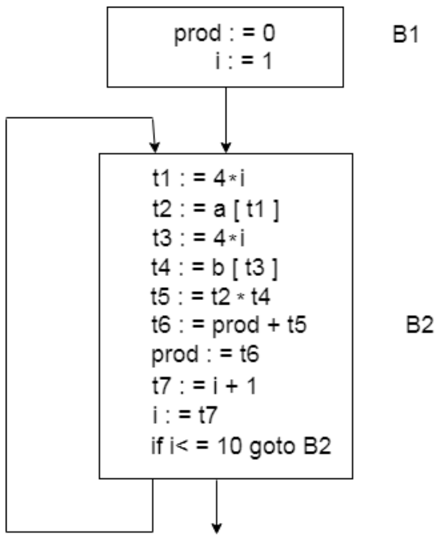
A three address statement x:= y+z is said to define x and to use y and z. A name in a basic block is said to be live at a given point if its value is used after that point in the program, perhaps in another basic block.

Flow Graph

Flow graph is a directed graph. It contains the flow of control information for the set of basic block.

A control flow graph is used to depict that how the program control is being parsed among the blocks. It is useful in the loop optimization.

Flow graph for the vector dot product is given as follows:

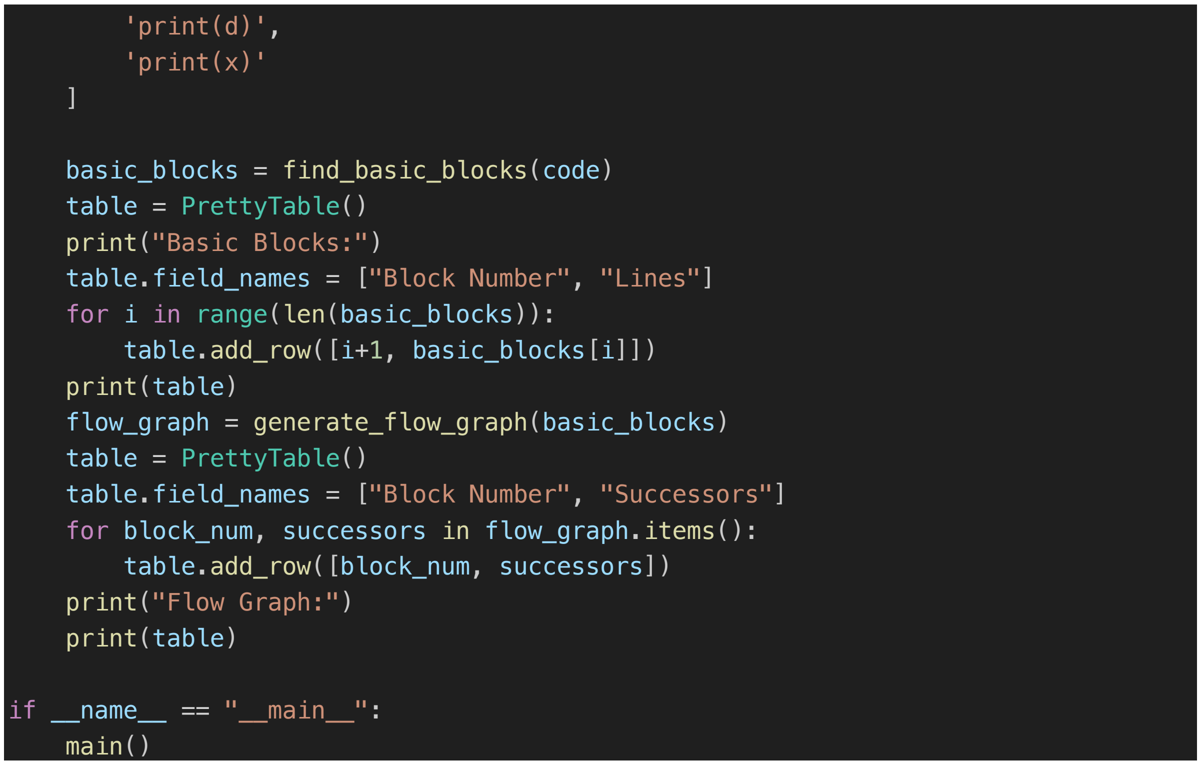


- Block B1 is the initial node. Block B2 immediately follows B1, so from B2 to B1 there is an edge.
- The target of jump from last statement of B1 is the first statement B2, so from B1 to B2 there is an edge.
- B2 is a successor of B1 and B1 is the predecessor of B2.

Implementation/ Code

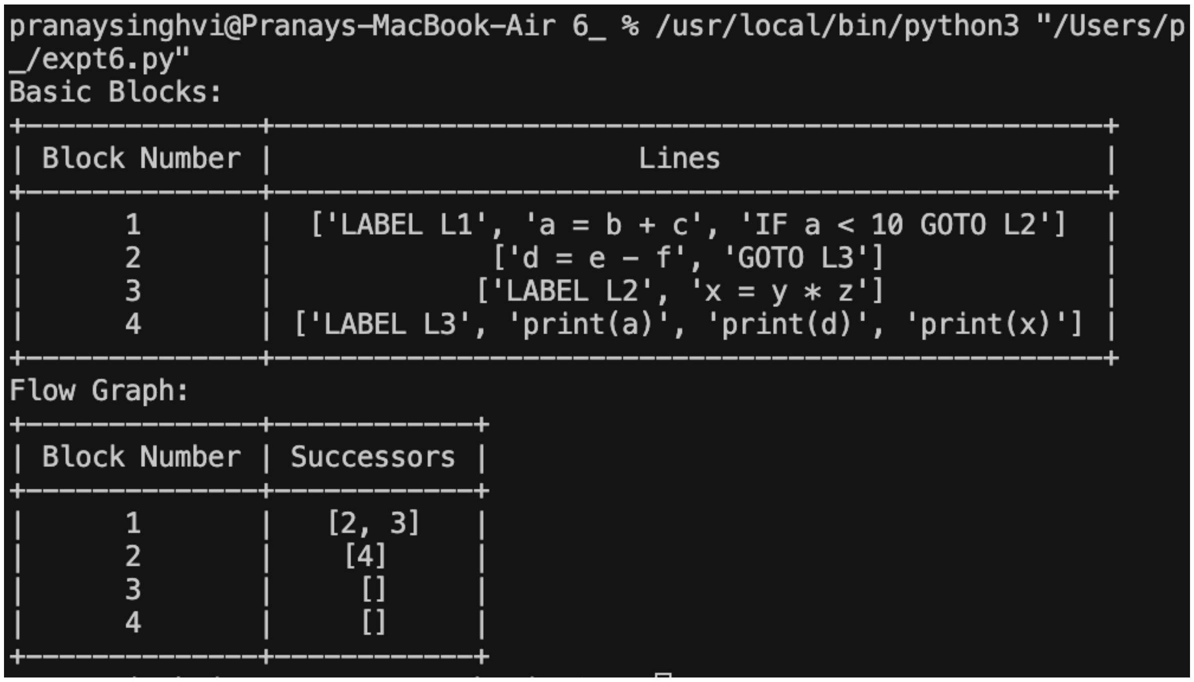
```python
from prettytable import PrettyTable
def find_basic_blocks(code):
    basic_blocks = []
    current_block = []

    for line in range(len(code)):
        if code[line].startswith('LABEL'):
            if current_block:
                basic_blocks.append(current_block)
                current_block = []
            current_block.append(code[line])
        elif code[line-1].startswith('IF'):
            basic_blocks.append(current_block)
            current_block = [code[line]]
        else:
            current_block.append(code[line])

    if current_block:
        basic_blocks.append(current_block)

    return basic_blocks
```

```python
def generate_flow_graph(basic_blocks):
    flow_graph = {}
    block_number = 1  # Initialize block number counter
    for block_num, block in enumerate(basic_blocks):
        successors = []
        for line_num in range(len(block)):
            if 'GOTO' in block[line_num]:
                goto_block = block[line_num].split()[-1]
                goto_block_num = None
                for i, blk in enumerate(basic_blocks):
                    if blk[0].split()[1] == goto_block:
                        goto_block_num = i + 1
                        break
                if goto_block_num is not None:
                    successors.append(goto_block_num)
            elif block[-1].startswith('IF'):
                if block_num +2 not in successors:
                    successors.append(block_num + 2)
            elif block[line_num].startswith('IF') or (line_num > 0 and
block[line_num-1].startswith('IF')):
                conditions = block[line_num].split()[2:]
                for condition in conditions:
                    goto_block = condition.split(':')[1]
                    goto_block_num = None
                    for i, blk in enumerate(basic_blocks):
                        if blk[0].split()[1] == goto_block:
                            goto_block_num = i + 1
                            break
                    if goto_block_num is not None:
                        successors.append(goto_block_num)

        flow_graph[block_number] = successors
        block_number += 1  # Increment block number counter for the next block

    return flow_graph

def main():
    code = [
        'LABEL L1',
        'a = b + c',
        'IF a < 10 GOTO L2',
        'd = e - f',
        'GOTO L3',
        'LABEL L2',
        'x = y * z',
        'LABEL L3',
        'print(a)',
```



```
        'print(d)',
        'print(x)'
    ]

    basic_blocks = find_basic_blocks(code)
    table = PrettyTable()
    print("Basic Blocks:")
    table.field_names = ["Block Number", "Lines"]
    for i in range(len(basic_blocks)):
        table.add_row([i+1, basic_blocks[i]])
    print(table)
    flow_graph = generate_flow_graph(basic_blocks)
    table = PrettyTable()
    table.field_names = ["Block Number", "Successors"]
    for block_num, successors in flow_graph.items():
        table.add_row([block_num, successors])
    print("Flow Graph:")
    print(table)

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

Output



```
pranaysinghvi@Pranays-MacBook-Air 6_ % /usr/local/bin/python3 "/Users/p
_/expt6.py"
Basic Blocks:
+--------------+-------------------------------------------------------+
| Block Number |                         Lines                         |
+--------------+-------------------------------------------------------+
|      1       |   ['LABEL L1', 'a = b + c', 'IF a < 10 GOTO L2']      |
|      2       |             ['d = e - f', 'GOTO L3']                  |
|      3       |             ['LABEL L2', 'x = y * z']                 |
|      4       |  ['LABEL L3', 'print(a)', 'print(d)', 'print(x)']     |
+--------------+-------------------------------------------------------+
Flow Graph:
+--------------+------------+
| Block Number | Successors |
+--------------+------------+
|      1       |   [2, 3]   |
|      2       |    [4]     |
|      3       |     []     |
|      4       |     []     |
+--------------+------------+
```

Conclusion

Through this experiment, I've learned to identify basic blocks and labels within code, understanding control flow.

References	[1] Geeksforgeeks. (September 21, 2023). Basic Blocks in Compiler Design https://www.geeksforgeeks.org/basic-blocks-in-compiler-design/ [2] Javatpoint. Flow Graph https://www.javatpoint.com/flow-graph [3] ChatGPT (April 5, 2024) Basic Block https://chat.openai.com/share/a93362bf-50b1-43f9-9427-7625500b6ef1