Course Name - Semester Year
Control Flow in Programs
Lab Exam - Set 3

Instructor: Firstname Lastname, Anonymous Institute, Country
`firstname.lastname@institute.email`

Month DD, YYYY

Roll Number: _____

Instructions

1. Please **refrain from engaging in any form of malpractice**, such as copying answers from any external sources.
2. *Please do **not** write your name or any other personal identification information anywhere on this sheet.*

Consent to use your work for academic research

These lectures and examination are part of an ongoing research that the instructor and his students are engaged in at Anonymous Institute, Country. Any personal information will be used only for grading and will not be disclosed elsewhere. If this data is used in any publication, it will be anonymised and aggregated.

Your participation in this study is completely voluntary. If you do not wish to allow the use of your data in this research, please tick the "NO" box below. In that case the data from your examination will not be used as part of the research. Otherwise, please tick "YES".

☐ **YES**, I consent to the use of my examination data for the purpose of academic research being conducted at Anonymous Institute, Country.

☐ **NO**, I do not consent the use of my examination data for the purpose of academic research being conducted at Anonymous Institute, Country.

Information about the examination

Exam Format

This lab exam consists of **five sections**. Each section contains the following:

1. **Program:** Statements with the corresponding locations listed to the left.
2. **Space for answers:** Space for you to write and draw the answers.

Space for **rough work** is provided in the last page.

Answer Format

In each section, your answer should include the following:

1. Structural Abstraction of the program.
2. Table of Control Transfer Functions.
3. Control Flow Graph without Error Edges.
4. Structurally Feasible Executions.
5. Logically Feasible Executions.
6. Actual Execution for the given input.

Refer to page 3 for an example of the expected answer format.
Please stick to the answer format specified in the example.

Additional Notes

1. Assume that arithmetic operations on strings throw an error.
2. To reduce clutter when drawing Control Flow Graphs, don't draw the Error edges.
3. When tracing Structurally and Logically Feasible Executions which enter a loop, showing one iteration is enough.

Example Question and Answer

Program

```
0
  x = read()
1
  y = 10 // x
2
  # end
```

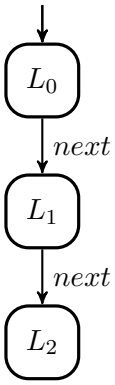
Structural Abstraction

```
0
  expression assignment
1
  expression assignment
2
  # end
```

Table of Control Transfer Functions

i	next	true	false	call	return	error
0	1					2
1	2					2
2						

Control Flow Graph



Structurally Feasible Executions

1. $L_0 \xrightarrow{next} L_1 \xrightarrow{next} L_2$
2. $L_0 \xrightarrow{next} L_1 \xrightarrow{error} L_2$
3. All other error executions (executions containing an error edge).

Note: Make sure to trace **at least one** structurally feasible execution that has an error edge.

Logically Feasible Executions

1. `x` is a number not equal to 0:
   $L_0 \xrightarrow{next} L_1 \xrightarrow{next} L_2$
2. `x` = 0:
   $L_0 \xrightarrow{next} L_1 \xrightarrow{error} L_2$
3. `x` is not a number:
   $L_0 \xrightarrow{next} L_1 \xrightarrow{error} L_2$

Actual Execution of the program for the input `x = 2`.

$$L_0 \xrightarrow{next} L_1 \xrightarrow{next} L_2$$

Section 1

Program

Write the structural abstraction of the program.

```
0
  q = 5
1
  p = read()
2
  m = p + q
3
  # end
```

Fill in the table of Control Transfer Functions with the appropriate locations for the program.

i	next	true	false	call	return	error
0						
1						
2						
3						

Draw the Control Flow Graph (CFG) for the program. To reduce clutter, don't draw the Error edges.

Trace the structurally feasible executions of the program.

Trace the logically feasible executions of the program.

Trace the actual execution of the program for the input `p = 5`.

Section 2

Program

Write the structural abstraction of the program.

```
0
a = 3
1
b = read()
2
if b > 5:
3
    a = b // a
4
else:
5
    a = b
6
c = a + b
7
# end
```

Fill in the table of Control Transfer Functions with the appropriate locations for the program.

i	next	true	false	call	return	error
0						
1						
2						
3						
4						
5						
6						
7						

Draw the Control Flow Graph (CFG) for the program. To reduce clutter, don't draw the Error edges.

Trace the structurally feasible executions of the program.

Trace the logically feasible executions of the program.

Trace the actual execution of the program for the input `b = 6`.

Section 3

Program

Write the structural abstraction of the program.

```
0
  p = read()
1
  c = p - 1
2
  while c != 0:
3
      y = p * c
4
      c = c - 1
5
      continue
6
  p = p + 1
7
  # end
```

Fill in the table of Control Transfer Functions with the appropriate locations for the program.

i	next	true	false	call	return	error
0						
1						
2						
3						
4						
5						
6						
7						

Draw the Control Flow Graph (CFG) for the program. To reduce clutter, don't draw the Error edges.

Trace the structurally feasible executions of the program. For executions which enter the while loop, showing one iteration is enough.

Trace the logically feasible executions of the program. For executions which enter the while loop, showing one iteration is enough.

Trace the actual execution of the program for the input `p = 3`.

Section 4

Program

Write the structural abstraction of the program.

```
0
def div(a, b):
1
    return a // b
2
x = read()
3
def func(p):
4
    p = p - 2
5
    k = div(5, p)
6
    return k
7
m = div(4, x)
8
n = func(x)
9
# end
```

Fill in the table of Control Transfer Functions with the appropriate locations for the program.

i	next	true	false	call	return	error
0						
1						
2						
3						
4						
5						
6						
7						
8						
9						

Draw the Control Flow Graph (CFG) for the program. To reduce clutter, don't draw the Error edges.

Trace the structurally feasible executions of the program.

Trace the logically feasible executions of the program.

Trace the actual execution of the program for the input `x = 2`.

Section 5

Program **Write the structural abstraction of the program.**

```
0
def func(y):
1
    return y + 1
2
p = read()
3
while p < 2:
4
    p = func(p)
5
    continue
6
p = p - 1
7
# end
```

Fill in the table of Control Transfer Functions with the appropriate locations for the program.

i	next	true	false	call	return	error
0						
1						
2						
3						
4						
5						
6						
7						

Draw the Control Flow Graph (CFG) for the program. To reduce clutter, don't draw the Error edges.

Trace the structurally feasible executions of the program. For executions which enter the while loop, showing one iteration is enough.

Trace the logically feasible executions of the program. For executions which enter the while loop, showing one iteration is enough.

Trace the actual execution of the program for the input `p = 0`.

Space for Rough Work