**Assignment 2 (20 points), SE 421, 8/24/2020, due: Wednesday, 9/2/2020**

**Name (Last, First): Stamatios Morellas**

**Electronic Copy Requirement**: (a) The answers should be typed. (b) The first page should include the top two lines with your last and the first name. (c) Include each question along with your answer to the question. (d) The file should be named HW2-lastname-firstname.

**Problem 1 (3 points):** What does the following program produce for each of the given inputs? In each case state which of the conditions in lines 3 through 9 would be TRUE when the program is executed with the given input.

```
 1  Read(X);
 2  N = 0;
 3  IF (X-N ≥ 60) then N = N + 60;
 4  IF (X-N ≥ 30) then N = N + 30;
 5  IF (X-N ≥ 15) then N = N + 15;
 6  IF (X-N ≥  8) then N = N +  8;
 7  IF (X-N ≥  4) then N = N +  4;
 8  IF (X-N ≥  2) then N = N +  2;
 9  IF (X-N ≥  1) then N = N +  1;
10  Result = N
```

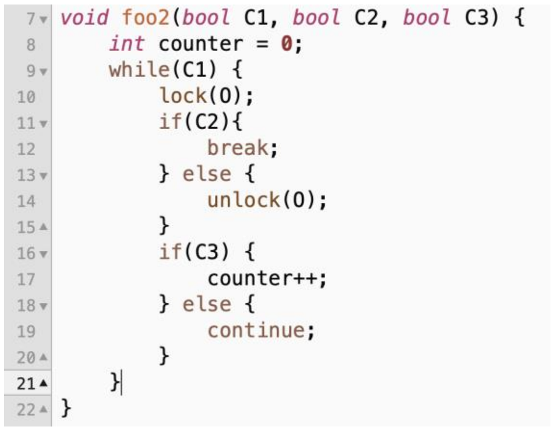
| Input X= | -5 | 1 | 75 | 120 | 125 |
|---|---|---|---|---|---|
| Result N = | **0** | **1** | **75** | **120** | **120** |
| True Conditions | **none** | **9** | **3, 5** | **3, 4, 5, 6, 7, 8, 9** | **3, 4, 5, 6, 7, 8, 9** |

**Problem 2 (5 points):** Fill in the Table (on next page) to describe *all* the remaining infeasible paths for the given program. In the given table, fill only as many blank rows of the as the number of remaining infeasible paths.

```
 1  Read(X);
 2  N = 0;
 3  IF (X-N ≥ 60) then N = N + 60;
 4  IF (X-N ≥ 30) then N = N + 30;
 5  IF (X-N ≥ 15) then N = N + 15;
 6  IF (X-N ≥  8) then N = N +  8;
 7  IF (X-N ≥  4) then N = N +  4;
 8  IF (X-N ≥  2) then N = N +  2;
 9  IF (X-N ≥  1) then N = N +  1;
10  Result = N
```

| Path | C1 | C2 | C3 | C4 | C5 | C6 | C7 | Result |
|---|---|---|---|---|---|---|---|---|
| | $X - N \geq 60$ | $X - N \geq 30$ | $X - N \geq 15$ | $X - N \geq 8$ | $X - N \geq 4$ | $X - N \geq 2$ | $X - N \geq 1$ | Feasible |
| 0001111 | F | F | F | T | T | T | T | No |
| 1011111 | T | F | T | T | T | T | T | No |
| 0101111 | F | T | F | T | T | T | T | No |
| 0000110 | F | F | F | F | T | T | F | No |
| 0011111 | F | F | T | T | T | T | T | No |
| 1111110 | T | T | T | T | T | T | F | No |
| 1000001 | T | F | F | F | F | F | T | No |

**Problem 3 (3 points):** Is *lock* (L) always followed by *unlock* (U) when the following code executes? List which of the behaviors listed in the Table can occur. Notation: $(E)^+$ means E repeated one or more times. $(E)^*$ means E repeated zero or more times.



```
void foo2(bool C1, bool C2, bool C3) {
    int counter = 0;
    while(C1) {
        lock(0);
        if(C2){
            break;
        } else {
            unlock(0);
        }
        if(C3) {
            counter++;
        } else {
            continue;
        }
    }
}
```

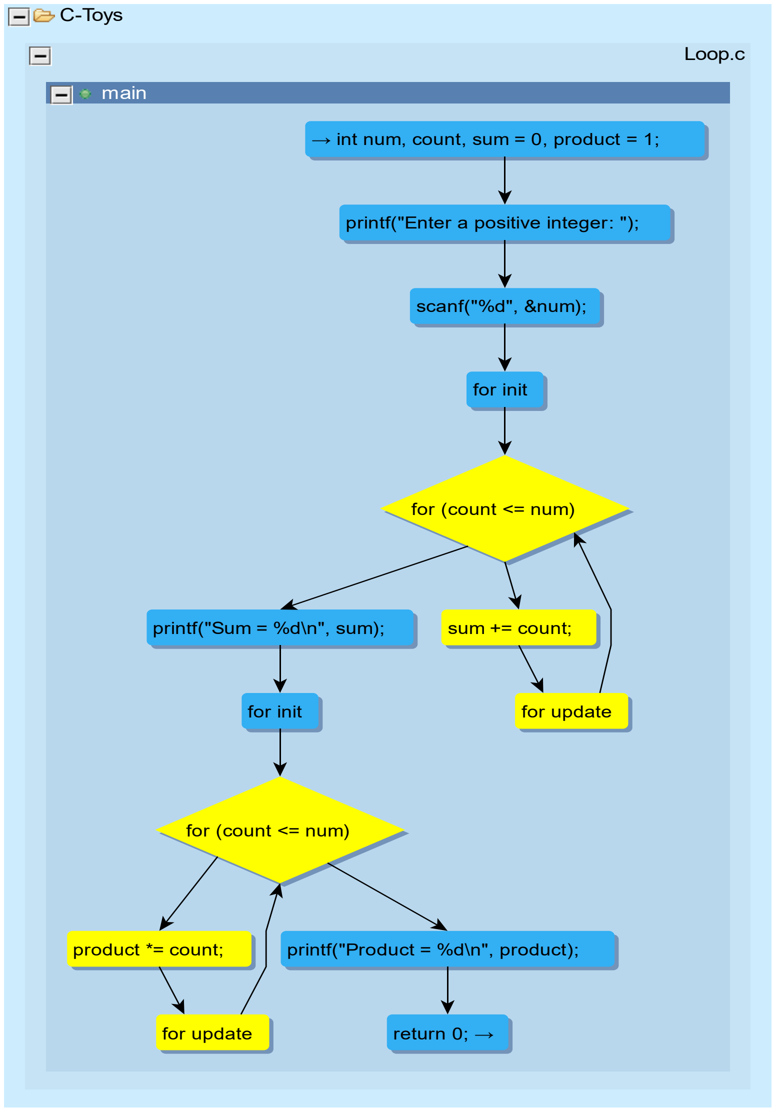
| Behavior | Yes or No |
|---|---|
| $(LU)^*$ | **No** |
| $(L)^+$ | **No** |
| UL | **No** |

**Problem 4 (9 points): Read the following information carefully and then answer the questions that follow.**

The *loop body* is the set of statements that execute as a part of the loop. The loop body includes *loop header* which is the unique entry point of the loop. Read the following illustration carefully before you answer the questions. Review the XINU code for the relevant functions.

```
int main() {
        // Loop Body is in bold typeface
        int num, count, sum = 0, product = 1;
        printf("Enter a positive integer: ");
        scanf("%d", &num);
        for(count = 1; count <= num; ++count) {
                sum += count;
        }
        printf("Sum = %d\n", sum);
        for(count = 1;count <=num; ++ count) {
               product *=count;
        }
        printf("Product = %d\n", product);
        return 0;
}
```

The Control Flow Graph (CFG) is shown next.



The following Atlas code segments attempt to extract the loop bodies from the CFG. You are expected to think if each of the attempts would work or not and why. Hopefully, there is no syntax error. Correct syntax error if you find one. The question is about the semantics and not the syntax.

**Segment 1**:

```
var dskenq = functions("dskenq")
var dskenqCFG = cfg(dskenq)
var dskenqLoops = dskenqCFG.nodes(XCSG.Loop)
var loopBody = edges(XCSG.ControlFlow_Edge).between(dskenqLoops,dskenqLoops)
show(loopBody)
```

**Segment 2:**

```
var kputc = functions("kputc")
var kputcCFG = cfg(kputc)
var kputcLoops = kputcCFG.nodes(XCSG.Loop)
var loopBody = edges(XCSG.ControlFlow_Edge).between(kputcLoops,kputcLoops)
show(loopBody)
```

**Segment 3:**

```
var kputc = functions("kputc")
var kputcCFG = cfg(kputc)
var kputcLoops = kputcCFG.nodes(XCSG.Loop)
var loopMembers = edges(XCSG.LoopChild).forward(kputcLoops).retainNodes
var loopBody = loopMembers.induce(edges(XCSG.ControlFlow_Edge))
show(loopBody)
```

1. Review the XINU code for the functions `dskenq` and `kputc`. Give the number of loops in each function. State how your count accounts for nested loops if any.

   Answer: Total number of loops in `dskenq` (according to segment 1 diagram): **1 total**, Total number of loops in `kputc` (according to segments 2 and 3): **5 total, 1 nested**

2. Show the result produced by each Atlas code segment. (Save and include the result of `show()`)

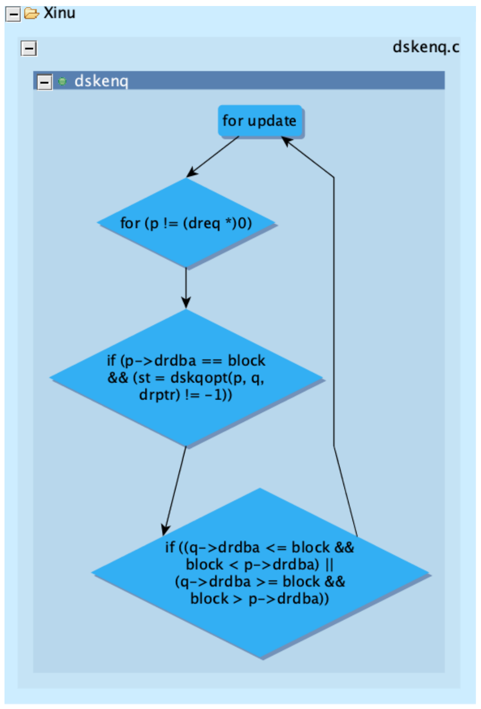
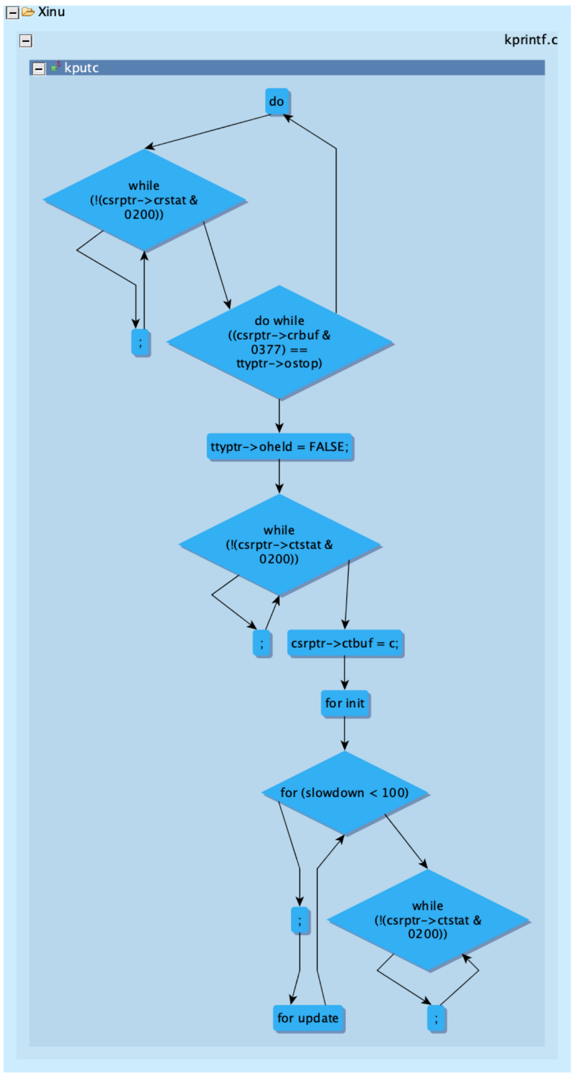
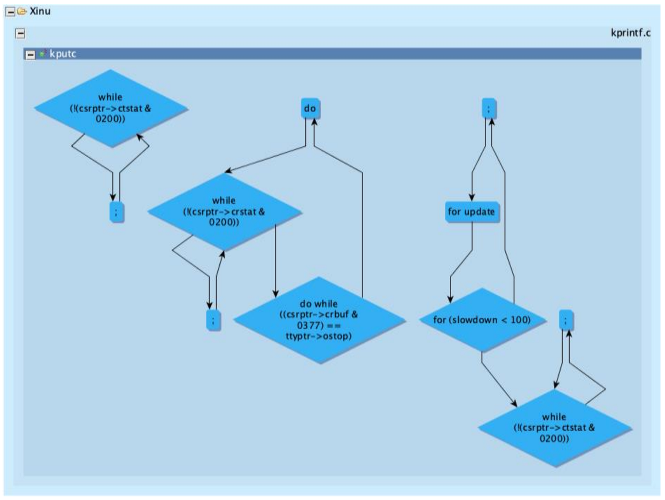

Xinu
dskenq.c
dskenq
for update
for (p != (dreq *)0)
if (p->drdba == block && (st = dskqopt(p, q, drptr) != -1))
if ((q->drdba <= block && block < p->drdba) || (q->drdba >= block && block > p->drdba))

*Figure 1: Segment 1 shell execution result*



Xinu
kprintf.c
kputc
do
while (!(csrptr->crstat & 0200))
;
do while ((csrptr->crbuf & 0377) == ttyptr->ostop)
ttyptr->oheld = FALSE;
while (!(csrptr->ctstat & 0200))
;
csrptr->ctbuf = c;
for init
for (slowdown < 100)
;
while (!(csrptr->ctstat & 0200))
for update
;

*Figure 2: Segment 2 shell execution result*



Xinu
kprintf.c
kputc
while (!(csrptr->ctstat & 0200))
do
;
;
while (!(csrptr->crstat & 0200))
for update
;
do while ((csrptr->crbuf & 0377) == ttyptr->ostop)
for (slowdown < 100)
;
while (!(csrptr->ctstat & 0200))

*Figure 3: Segment 3 shell execution result*

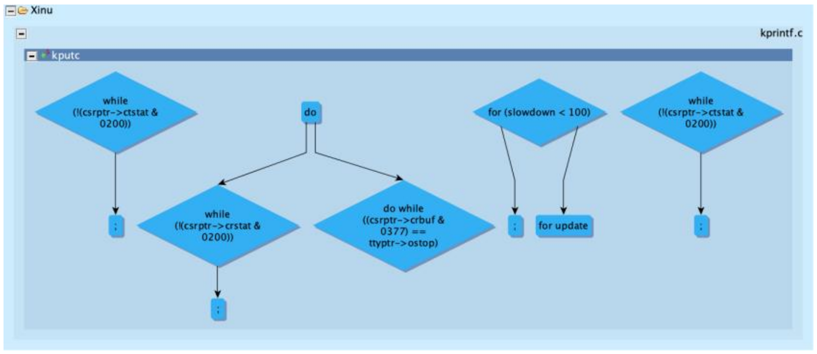

Xinu
kprintf.c
kputc
while (!(csrptr->ctstat & 0200))
;
do
while (!(csrptr->crstat & 0200))
;
do while ((csrptr->crbuf & 0377) == ttyptr->ostop)
for (slowdown < 100)
;
for update
while (!(csrptr->ctstat & 0200))
;

*Figure 4: Segment 4 shell execution result*

3. Is there a difference between Segment 1 and 2? Precise answer in one sentence please.

   Answer: The syntax of Segments 1 and 2 are the *same* in terms of the functions they perform, but their outputs are *different*.

4. Summarize whether the loop bodies are computed correctly by choosing one of the three: (a) Segment 1 computes it correctly for `dskenq`, (b) Segment 2 computes it correctly for `kputc`, or (c) Segments 1 and 2 compute it correctly for both the functions.

   Answer: (c) Segments 1 and 2 compute the correct CFG for both `dskenq` and `kputc`

5. Compare the results of segments 2 and 3. Which of them computes the loop bodies correctly? (a) Segment 2, (b) Segment 3, or (c) Both.

   Answer: (a) Segment 2 seems to have a more accurate computation of the loop body.

6. Experiment with the following code segment. Try it on: (a) functions with one loop, (b) functions with more than one loop. What do you learn from this experiment? Precisely in at most three short sentences please.

**Segment 4:**

```
var kputc = functions("kputc")
var kputcCFG = cfg(kputc)
var kputcLoops = kputcCFG.nodes(XCSG.Loop)
var loopMembers = edges(XCSG.LoopChild).forward(kputcLoops)
show(loopMembers)
```

Answer: The two functions I tried Segment 4 on are `qxdump` and `pxdump`. The CFG results are displayed below. Based on what I saw, the CFG's looked rather similar for each of the different functions. By using this segment, it was easier for me to identify how many loops are in each function since the loopback edges are not shown in these CFG renderings.

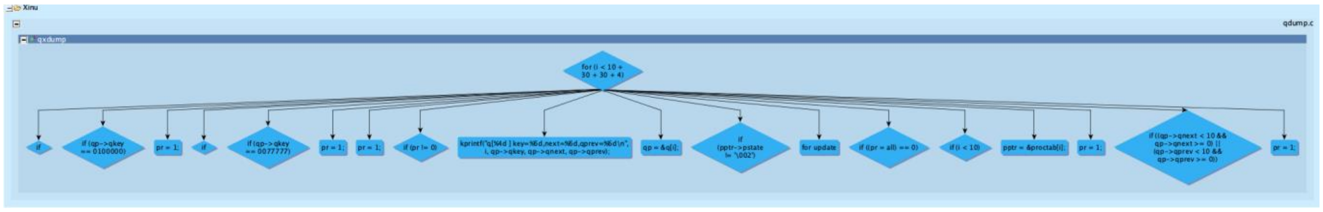

Xinu
qdump.c
qxdump
for (i < 10 + 30 + 30 + 4)
if
if (qp->qkey == 0100000)
pr = 1;
if
if (qp->qkey == 0077777)
pr = 1;
pr = 1;
if (pr != 0)
kprintf("q[%4d ] key=%6d,next=%6d,qprev=%6d\n", i, qp->qkey, qp->qnext, qp->qprev);
qp = &q[i];
if (pptr->pstate != '\002')
for update
if ((pr = all) == 0)
if (i < 10)
pptr = &proctab[i];
pr = 1;
if ((qp->qnext < 10 && qp->qnext >= 0) || (qp->qprev < 10 && qp->qprev >= 0))
pr = 1;

*Figure 5: CFG output for `qxdump` function*

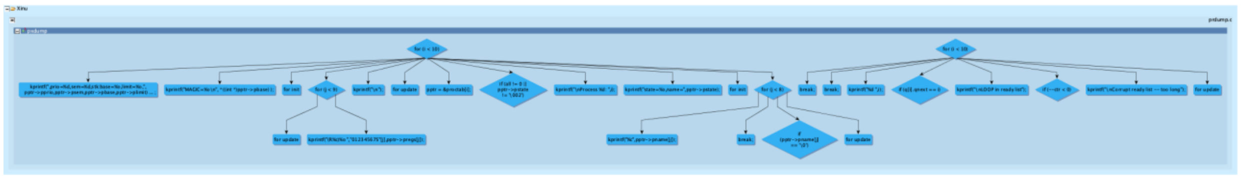

Xinu
pxdump.c
pxdump
for (i < 10)
kprintf("MAGIC=%o\n", *((int *)pptr->pbase) );
for init
for (j < 9)
kprintf("\n");
for update
pptr = &proctab[i];
if (all != 0 || pptr->pstate != '\002')
kprintf("\nProcess %d: ",i);
kprintf("state=%o,name=",pptr->pstate);
for init
for (j < 8)
for update
kprintf("%c",pptr->pname[j]);
break;
if (pptr->pname[j] == '\0')
for update
for (i < 10)
break;
break;
kprintf("%d ",i);
if (q[i].qnext == 0)
kprintf("\nLOOP in ready list");
if (--ctr < 0)
kprintf("\nCorrupt ready list --- too long");
for update

*Figure 6: CFG output for* `pxdump` *function*