SMITH COLLEGE

Spring 2021 FINAL EXAM

Course Number and Section: CSC 151

Course Title: Introduction to Computer Graphics

Instructor: Nicholas Howe

Exam Format: Take-home, Open book

STUDENT NAME:
ID NUMBER:
CLASS YEAR:

ACADEMIC HONOR CODE

Students and faculty at Smith are part of an academic community defined by its commitment to scholarship, which depends on scrupulous and attentive acknowledgement of all sources of information and honest and respectful use of college resources.

Smith College expects all students to be honest and committed to the principles of academic and intellectual integrity in their preparation and submission of course work and examinations. All submitted work of any kind must be the original work of the student who must cite all the sources used in its preparation.

Student Signature: _____

EXAM INSTRUCTIONS

ALL ANSWERS SHOULD BE SCANNED AND SUBMITTED ON MOODLE. PLEASE INCLUDE THE SIGNED AND SCANNED HONOR CODE STATEMENT FROM THE FRONT PAGE OF THIS EXAM.

THIS IS AN OPEN-BOOK ASSESSMENT. YOU MAY USE THE TEXTBOOK, CLASS NOTES, AND AND (WITH A CITATION) INTERNET REFERENCES. YOU MAY NOT DISCUSS ANY PART OF THIS EXAM WITH ANY PERSON OTHER THAN THE INSTRUCTOR, ELECTRONICALLY OR OTHERWISE, EXCEPT AS BELOW.

Option A. *Take the exam on your own, without any assistance from others. Under this option, the exam is due on May 21 at 4:00 PM. (All deadlines for this assignment are EDT.)*

Option B. *Participate in a group exam process. Under this option, you must turn in your individual effort on the exam by Wednesday, May 19 at 4:00 PM. I will then assign you to a group, and groups will have until Thursday, May 20 at 4:00 PM to submit a single revised solution. Finally, the entire set of individuals who have chosen Option B will collaborate to produce their best combined solution by Friday, May 21 at 4:00 PM.*

Vocabulary *(20 points)*

Give brief (one or two sentences) definitions for the terms or concepts below.

a.) Abstract stack machine
b.) Garbage collector
c.) Mutable reference
d.) Immutable reference
e.) C preprocessor
f.) Subtype
g.) Class
h.) Script
i.) Variable binding
j.) Concurrency

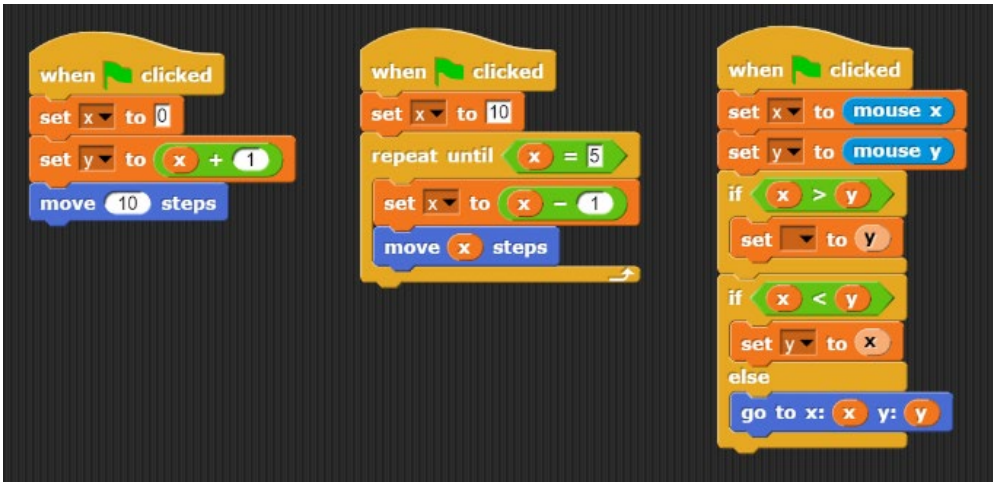
Program Execution *(12 points)*

Given a program written in a high level language and spread over multiple files, describe the steps involved in transforming it to a form that can actually be run on the CPU. For each transformation, name the entity that performs the transformation and describe when it happens.

a.) The program is written in C
b.) The program is written in Python
c.) The program is written in Java

Language Specification (16 points)

Write a concise set of Backus-Naur rules that can generate the subset of Snap! shown in the image below. You don't need to write rules that generate numbers or variable names; just leave these as non-terminals. Don't include other operators and structures not shown in the image.



Type Inference *(16 points)*

Write the type of each of the following OCaml expressions in the first blank provided, or *ill-typed* if the expression does not type check. Assume that the following are already defined:

```
let x (a: 'a list) (b: 'b list) (c: 'a -> 'b -> 'c) : 'c list = (* redacted *)
let y (a: int) (b: int) : int = (* redacted *)
let z (a: 'a) (b: 'a list) : 'a list = (* redacted *)
```

The first has been done for you as an example.

```
let a : ____int_____ = y 2 3
let b : _____________ = z a []
let c : _____________ = z a
let d : _____________ = c b
let e : _____________ = z b []
let f : _____________ = x b b y
let g : _____________ = x [1;2] [3;4] y
let h : _____________ = z z [z]
let i : _____________ = fun (r) (s) (t) -> x s t r
```

Loops & Recursion *(24 points)*

For each example below, determine whether the two functions always produce an equivalent result for any input. If they do not, identify a set of inputs inputs where the result would differ.

```
int fun1(int* a, int n) {
    int s = 0;
    for (int i = 0; i < n; i++) {
        s += a[i];
    }
    return s;
}
```

```
int fun2(int* a, int n) {
    int s = a[0];
    for (int i = 1; i < n; i++) {
        s += a[i];
    }
    return s;
}
```

```
int fun3(int* a, int n) {
    int s = 0, i = 0;
    while (i < n) {
        i++;
        s += a[i];
    }
    return s;
}
```

```
int fun4(int* a, int n) {
    int s = 0, i = 0;
    do {
        s += a[i];
        i++;
    } while (i < n);
    return s;
}
```

```
int fun5(int* a, int n) {
    if (n == 0) {
        return 0;
    } else {
        return *a+fun5(a+1,n-1);
    }
}
```

```
int fun6(int* a, int n) {
    if (n == 1) {
        return a[0];
    } else {
        return *a+fun6(a+1,n-1);
    }
}
```

```
int fun7(int* a, int n) {
    int s = 0;
    for (int i = 0; i < n-1; i++) {
        s += a[i];
    }
    return s+a[n-1];
}
```

a.) **fun1** and **fun2**.
b.) **fun1** and **fun3**.
c.) **fun2** and **fun4**.
d.) **fun2** and **fun6**.
e.) **fun1** and **fun5**.
f.) **fun4** and **fun7**.
g.) **fun5** and **fun6**.
h.) **fun1** and **fun7**.

Scope *(12 points)*

For each program, identify all identifiers that are bound to a value in the current scope when the program reaches the line identified as HERE.

```
let f (b: int): int =
  let c = (b-1)*(b+1) in
    (* HERE - OCaml *)
    b*b-c

let d = let a = 7 in f a
```

```
int f(int b) {
    int c = (b-1)*(b+1);
    // HERE - C
    return b*b-c;
}

int main() {
    int a = 7;
    int d = f(a);
}
```

```
public class Scope {
    static int e = 3;
    int f = 5;

    public static int f(int b) {
        int c = (b-1)*(b+1);
        // HERE - Java
        return b*b-c;
    }

    public static void main(String[] args) {
        int a = 7;
        int d = f(a);
    }
}
```

a.) OCaml
b.) C
c.) Java