Average raw grade on this final was 69/100. A = 79, B = 69, C = 59, D = 49

SMITH COLLEGE

Spring 2021 FINAL EXAM

Course Number and Section: CSC 151

Course Title: Introduction to Programming Languages

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 ANY RESOURCES LINKED FROM THE COURSE WEB PAGE. 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.*

Average raw grade on this final was 69/100. A = 79, B = 69, C = 59, D = 49

Vocabulary

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

a.) Abstract stack machine *A model of computation designed to resolve variable references with mutability.*
b.) Garbage collector *A service provided by some languages that handles release of memory that is no longer in use.*
c.) Mutable reference *Reference to a stored value; the value referred to may change as the program executes*
d.) Immutable reference *Reference to a stored value that will never change as the program executes*
e.) C preprocessor *Before the program is actually compiled, the C preprocessor modifies the code according to directives contained within it.*
f.) Subtype *A type that can fulfill all the requirements of some supertype*
g.) Class *A group of objects that share a common description, or the description itself.*
h.) Script *Computer code that is interpreted, often used as part of some larger context.*
i.) Variable binding *The process of associating a program identifier with a value*
j.) Concurrency *A state of affairs where multiple threads of execution are executing at once, interleaving their actions.*

Program Execution

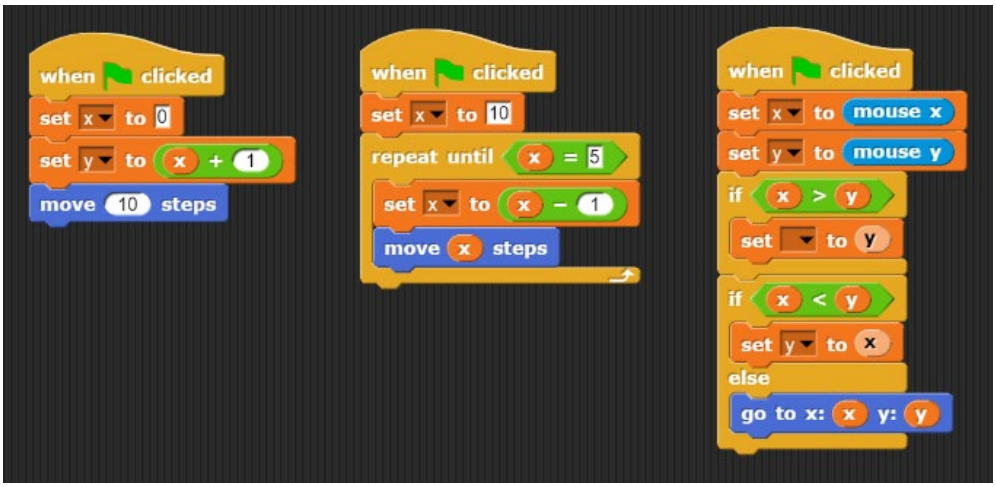
Given a program written in a high level language and spread over multiple files, describe the steps involved in transforming it to a form that can actually be run on the CPU. For each transformation, name the entity that performs the transformation and describe when it happens.

a.) The program is written in C
The preprocessor transforms the high-level code. Then the compiler converts each file to a machine code object file. Finally the linker produces a single executable file. All this happens before the program runs.
b.) The program is written in Python
Nothing happens before the program is run. When the program is run, the high-level code is converted implicitly to Python bytecode. The interpreter then converts the bytecode to machine code as necessary to execute it.
c.) The program is written in Java
The compiler converts each file to Java bytecode before the program is run. As the program runs, the Java virtual machine retrieves the next line of bytecode from the currently running file and converts it to machine code that can be executed.

Average raw grade on this final was 69/100. A = 79, B = 69, C = 59, D = 49

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.



*<program> := **when clicked** <block>**

<block> := <action>|<repeat>|<if>

*<repeat> := **repeat until** <bool> <block>**

*<if> := **if** <bool> <block>* [**else** <block*>]*

*<action> := **set** <var> **to** <val> | **move** <val> **steps** | **go to x:** <val> **y:** <val>*

*<bool> := <val> **<** <val> | <val> **=** <val> | <val> **>** <val>*

<val> = <number> | <variable> | <expr>

*<expr> := <val> **+** <val> | <val> **-** <val>*

Average raw grade on this final was 69/100. A = 79, B = 69, C = 59, D = 49

Type Inference

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 f (a: 'a list) (b: 'b list) (c: 'a -> 'b -> 'c) : 'c list = (* redacted *)
let g (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 : _____int list_____ = z a []
let c : _____int list -> int list_____ = z a
let d : _____int list_____ = c b
let e : _____int list list______ = z b []
let f : _____int list_____ = x b b y
let g : _____int list_____ = x [1;2] [3;4] y
let h : (int list -> int list) list = z z [z]
let i : ('a -> 'b -> 'c) -> 'a list -> 'b list -> 'c list = fun (r) (s) (t) -> x s t r
```

Average raw grade on this final was 69/100. A = 79, B = 69, C = 59, D = 49

Loops & Recursion

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+fun2(a+1,n-1);
    }
}
```

```
int fun6(int* a, int n) {
    if (n == 1) {
        return a[0];
    } else {
        return *a+fun2(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**. *Different when n = 0*
b.) **fun1** and **fun3**. *Different*
c.) **fun2** and **fun4**. *The same*
d.) **fun2** and **fun6**. *Different (fun6 won't terminate when n = 0)*
e.) **fun1** and **fun5**. *The same*
f.) **fun4** and **fun7**. *Different when n = 0*
g.) **fun5** and **fun6**. *Different (fun6 won't terminate when n = 0)*
h.) **fun1** and **fun7**. *Different when n = 0*

Average raw grade on this final was 69/100. A = 79, B = 69, C = 59, D = 49

Scope

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 *a, b, c*

b.) C *b, c*

c.) Java *b, c, e, f*