Welcome to the Programming Languages Final Exam

Welcome to the Programming Languages Final Exam.

This exam is open-book, open-notes, open-internet, open-everything-else-but-not-other-people-or-communication-with-people. You may use any resource you find useful, including any notes you have taken during the course, any textbook you have, any other notes you have, any other textbook you have, any other resource you find on the internet, and any other resource you find anywhere else. The only exceptions are that you may not communicate with other people about the exam, and you may not use any resource that you know or believe was created by a person who has taken this course in the past or is currently taking this course.

Please read the rules carefully, and be sure you understand them. If you have any questions, please contact the TA.

Note: If you have trouble accessing the Gradescope site, please contact the TA.

New submission for: Final Exam

Due: 12/16/2020 @ 02:29:59 AM EDT

Honor Pledge

On my honor, I have neither given nor received any unauthorized aid on this exam.

By typing your first and last name in the space provided below you are electronically signing or initials then signing the following statement:

- I have not given or received any unauthorized aid on this exam.
- You will submit your response on Gradescope by the deadline.
- You will submit your response no later than 100 minutes from the time you first accessed the exercise of the exam on Gradescope, or you will be assigned a penalty equivalent to the amount of time you spend over 100 minutes.
- You will follow the submission instructions for this exercise.

If you feel you will cause stress or a panic attack, you may wish not to proceed and you will receive a grade of zero.

Write your answer here:
Honor Pledge

Save

Question 1. (1 pt) Consider the parse tree below. All terminal symbols are on the same level, in the order shown.

(1 pt) Choose the number of the grammar in question 2 that this parse tree

Write your answer below:
Enter a number between 1 and 4 (inclusive) here:
num = 1 | num = 2 | num = 3 | num = 4 | num

Save

(1 pt) Choose the grammar in question 2 that matches this parse tree

Write your answer below:
Enter a number between 1 and 4 (inclusive) here:
num = 1 | num = 2 | num = 3 | num = 4 | num

Save

(1 pt) Consider the following Rust code:

```
fn f(x:i32) -> i32 {
  if x == 0 { 0 }
  else { x + f(x - 1) }
}

fn g(x:i32) -> i32 {
  if x == 0 {
    0
  }
  else {
    x + g(x - 1)
  }
}

fn h(x:i32) -> i32 {
  if x == 0 {
    0
  }
  else {
    x + h(x - 1)
  }
}

print("f(5) = ", f(5));
print("g(5) = ", g(5));
print("h(5) = ", h(5));
```

Write your answer here:
Enter a number between 1 and 4 (inclusive) here:
num = 1 | num = 2 | num = 3 | num = 4 | num

Save

(1 pt) Consider the following Rust code:

```
fn f(x:i32) -> i32 {
  if x == 0 {
    0
  }
  else {
    x + f(x - 1)
  }
}

fn g(x:i32) -> i32 {
  if x == 0 {
    0
  }
  else {
    x + g(x - 1)
  }
}

fn h(x:i32) -> i32 {
  if x == 0 {
    0
  }
  else {
    x + h(x - 1)
  }
}

print("f(5) = ", f(5));
print("g(5) = ", g(5));
print("h(5) = ", h(5));
```

Write your answer here:
Enter a number between 1 and 4 (inclusive) here:
num = 1 | num = 2 | num = 3 | num = 4 | num

Save

(1 pt) Consider the following Rust function:

```
fn f(
  x:i32,
  y:i32,
  z:i32,
  w:i32,
) {
  if x == 0 {
    print("x = 0\n");
    if y == 0 {
      print("y = 0\n");
      if z == 0 {
        print("z = 0\n");
        if w == 0 {
          print("w = 0\n");
        }
      }
    }
  }
}
```

Write your answer here:
Enter a number between 1 and 4 (inclusive) here:
num = 1 | num = 2 | num = 3 | num = 4 | num

Save

(1 pt) Consider the following Rust function:

```
fn f(
  x:i32,
  y:i32,
  z:i32,
  w:i32,
) {
  if x == 0 {
    print("x = 0\n");
    if y == 0 {
      print("y = 0\n");
      if z == 0 {
        print("z = 0\n");
        if w == 0 {
          print("w = 0\n");
        }
      }
    }
  }
}
```

Write your answer here:
Enter a number between 1 and 4 (inclusive) here:
num = 1 | num = 2 | num = 3 | num = 4 | num

Save

(1 pt) Consider the following Rust code. Note that the alleged legal code being that is at least one parse error. Write down how many parse errors you think are there.

Write your answer below:
Enter a number between 1 and 4 (inclusive) here:
num = 1 | num = 2 | num = 3 | num = 4 | num

Save

(1 pt) Consider the following Rust code. Note that the alleged legal code being that is at least one parse error. Write down how many parse errors you think are there.

Write your answer below:
Enter a number between 1 and 4 (inclusive) here:
num = 1 | num = 2 | num = 3 | num = 4 | num

Save

(1 pt) Consider the following Rust code:

```
fn f(x:i32) -> i32 {
  if x == 0 {
    0
  }
  else {
    x + f(x - 1)
  }
}

fn g(x:i32) -> i32 {
  if x == 0 {
    0
  }
  else {
    x + g(x - 1)
  }
}

fn h(x:i32) -> i32 {
  if x == 0 {
    0
  }
  else {
    x + h(x - 1)
  }
}

print("f(5) = ", f(5));
print("g(5) = ", g(5));
print("h(5) = ", h(5));
```

Write your answer here:
Enter a number between 1 and 4 (inclusive) here:
num = 1 | num = 2 | num = 3 | num = 4 | num

Save

(1 pt) Consider the following Rust code:

```
fn f(x:i32) -> i32 {
  if x == 0 {
    0
  }
  else {
    x + f(x - 1)
  }
}

fn g(x:i32) -> i32 {
  if x == 0 {
    0
  }
  else {
    x + g(x - 1)
  }
}

fn h(x:i32) -> i32 {
  if x == 0 {
    0
  }
  else {
    x + h(x - 1)
  }
}

print("f(5) = ", f(5));
print("g(5) = ", g(5));
print("h(5) = ", h(5));
```

Write your answer here:
Enter a number between 1 and 4 (inclusive) here:
num = 1 | num = 2 | num = 3 | num = 4 | num

Save

(1 pt) Consider the following Rust function:

```
fn f(
  x:i32,
  y:i32,
  z:i32,
  w:i32,
) {
  if x == 0 {
    print("x = 0\n");
    if y == 0 {
      print("y = 0\n");
      if z == 0 {
        print("z = 0\n");
        if w == 0 {
          print("w = 0\n");
        }
      }
    }
  }
}
```