CMPSC 461: Programming Language Concepts, Spring 2024
Assignment 4 Practice Notes Packet

Prof. Suman Saha

October 7, 2025

Problem 1: Calling sequence

Consider the following C program :

```
1   void func1(void);
2   void func2(void);
3   void func3(void);
4
5   void main(){
6       int a,b,c;
7       /* body contents of main*/
8   }
9
10  void func1(void){
11      int b,c,d;
12      /* body contents of func1*/
13  }
14
15  void func2(void){
16      int c,d,e;
17      /* body contents of func2*/
18  }
19
20  void func3(void){
21      int d,e,f;
22      /* body contents of func3*/
23  }
```

Assuming dynamic scoping is used, what variables are visible during the execution of the last called function in the following sequences? Include the name of the function in which the variable was defined along with each visible variable.

1. **main** calls **func1**, **func1** calls **func2**, **func2** calls **func3**
2. **main** calls **func1**, **func1** calls **func3**
3. **main** calls **func2**, **func2** calls **func3**, **func3** calls **func1**
4. **main** calls **func3**, **func3** calls **func3**
5. **main** calls **func1**, **func1** calls **func3**, **func3** calls **func2**

For example, the sequence "**main** calls **func3**, **func3** calls **func2**, **func2** calls **func1**" will lead to the answer :
Visible variables : a(main), b(func1), c(func1), d(func1), e(func2), f(func3)

Problem 2: Parameter Passing 1

Observe the following C++ code that swaps two variables :

```
#include<stdio.h>
void swap(int n1, int n2)
{
  int temp = n1;
  n1 = n2;
  n2 = temp;

}
int main() {
    int x = 25;
    int y = 70;
    swap(x, y);
    printf("The numbers after swapping n1 and n2 %d %d \n",x, y);
    return 0;
}
```

How would the output of the program change if the parameters are :

1. Called by value?
2. Called by reference?

Problem 3: Parameter passing 2

Consider the following C-like program. Write down what will be printed out when the parameters are passed (1) by value, (2) by reference and (3) by value return. For each answer, briefly explain how you derived it.

```
1     int x=5, y=6;
2     void foo(int a, int b) {
3         x = a+b;
4         b = a+a;
5     }
6
7     main () {
8         foo(x,y);
9         print x, y;
10    }
```

Problem 4: Tail Recursion

Consider the following JAVA program

```
1 public int countNodes(Node<T> start) {
2     if (start == null) {
3         return 0;  // Base case
4     }
5     else {
6         return 1 + countNodes(start.next);
7     }
8 }
```

Write down a **tail recursive** implementation of the function **countNodes** in JAVA language or programming language of your choice. You may use a helper function in your solution.

Problem 5: Tail-recursion

The Collatz sequence for a number n is generated as follows:

- If n is even, divide it by 2.
- If n is odd, multiply it by 3 and add 1.
- The sequence meet its termination when it first encounters the number 1, as it goes into a short repetition $(1, 4, 2, 1, ...)$

Example: For $n = 6$, the Collatz sequence is: $6, 3, 10, 5, 16, 8, 4, 2, 1$. Therefore, we define the term **Collatz-length** as the length of the Collatz sequence with starting number n before it reaches 1. E.g., the Collatz-length of 6 is 8.

1. Write down a **tail recursive** implementation of the function **collatzlength** in python language preferably or programming language of your choice. You may use the helper function in your solution.
2. Explain why your implementation of Collatzlength is **tail-recursive**?

Problem 6: Tail Recursion I

Consider the following Python program

```
1  def sq_sum(lst):
2      """Sums a list of numbers squared."""
3      if len(lst) == 0:
4          return 0
5      return (lst[0] * lst[0]) + sum(lst[1:])
6
7  def division_mult(number):
8      ""Returns the multiplication of any number continuously halved until 1 or less""
9      temp = number / 2
10     if ((temp == 1) or (temp == 0)):
11         return 1
12     else:
13         return temp * division_mult(temp)
```

1. Write down a **tail recursive** implementation of the function **sq_sum** in python language or programming language of your choice. You may use the helper function in your solution.
2. Write down a **tail recursive** implementation of the function **division_mult** in python language or programming language of your choice. You may use the helper function in your solution.

Note: There are different ways to do achieve this.

Problem 7: Tail Recursion II

Consider the following Python program

```
1 def cube_sum(lst):
2     """Sums a list of numbers cubed."""
3     return cube(lst, 0)
4
5 def cube(lst, value):
6     if len(lst) == 0:
7         return value
8     temp = (lst[0] * list[0] * list[0]) + value
9     return cube(lst[1: ], temp)
```

1. Write down a **recursive** implementation of the function **cube_sum** in python language or programming language of your choice. You may use the helper function in your solution.

2. Draw the highest run time stack when the input is 1, 3, 9 for both the **tail recursive and recursive** implementation of the function **cube_sum**. No need to show links and assume global calls the function.

Problem 8: Exception handling

Consider the following code snippet with exceptions. Note that the `main` function calls `foo` in the nested `try` block.

```
1  void foo () {
2      try {
3          throw new Exception1();
4          print ("Firefly");
5          try {
6              throw new Exception1();
7              print ("Dream");
8                  try {
9                      throw new Exception2();
10                     print ("Song");
11                 }
12         }
13
14     }
15     catch(Exception1 e1) {
16         print "handler one";
17     }
18     print ("Edge");
19     throw new Exception2();
20 }
21 void main () {
22     try {
23         print ("Sam");
24         try {
25             print ("Meme");
26             foo();
27         }
28         catch(Exception1 e1) { print "handler two"; }
29         print ("Robin");
30     }
31     catch(Exception2 e2) { print "handler three"; }
32 }
```

a) (7 pt) Write down what will be the output of the following program and briefly justify your answer.

b) (7 pt) Instead of the "replacement" semantics of exception handling used in modern languages (i.e., the semantics introduced in lecture), a very early design of exception handling introduced in the PL/I language uses a "binding" semantics. In particular, the design dynamically tracks a sequence of "catch" blocks that are currently active; a catch block is active whenever the corresponding try block is active. Moreover, whenever an exception is thrown, the sequence of active "catch" blocks will be traversed (in a first-in-last-out manner) to find a matching handler. Furthermore, execution will resume at the statement following the one that throws exception, rather than the next statement after the matching "catch" block as we have seen in the "replacement" semantics.
What will be the output of the following program if the language uses the "binding" semantics? Briefly justify your answer.

Problem 9: Calling Sequence

enumerate

What are function parameters? Briefly explain with an example.

What are function arguments? Briefly explain with an example.

Name the five parameter passing mechanisms.

When passing large arrays/objects, which passing mechanism is the most efficient and why?

What is one downside of always using pass-by-reference?

Problem 10: Calling Sequence

enumerate

How are values passed in call by value (CBV)?

How are values passed in call by result (CBR)?

How are values passed in call by value-result (CBVR)?

How are values passed in call by reference (CBR - Reference)?

How are values passed in call by value (CBN)?

Problem 11: Exception Handling/Recursion Theory

Briefly explain the following:

1. What is the difference between pre-defined exceptions and user-defined exceptions?
2. List at least two examples of pre-defined exceptions.
3. What happens when a exception is not handled in a function call?
4. Why are local variables in recrusive functions dynamically allocated?
5. What are some benefits of using tail-recursion over traditional recursion?

Problem 12: Exception Handling/Recursion Theory

Answer True or False for the following statements and provide a brief explanation to each question:

1. The order in which catch blocks are listed is not important.
2. If an exception is thrown in a try block, the remaining statements in that try block are executed after executing a catch block.
3. You can have multiple catch blocks for a single try block.