CSE 340 SPRING 2019

HOMEWORK 6 SOLUTION

Due Wednesday, 4/24/2019 by 11:59 pm

- You should write your name on your submission.
- Remember that late submissions are not accepted for homework.
- **You should answer the questions in the order they are asked**
- **You should submit a single pdf file for the solution not multiple files.**

Problem 1. Consider the following code in C syntax:

```c
#include <stdio.h>

int temp = 0;

int sum(int i, int m, int n, int ai)
{
    for (i = m; i < n; i++) {
        temp = temp + ai;
    }
    return temp;
}

int main()
{
     int a[9] = {1,10,10,10,100,100,100,100,100};
     int b[4][4] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};

     int temp = 4;
     int m = 0;
     int n = 3;
     int i = 1;
     int j = 1;
     int result;

     result = sum(i, m , n, i);
     printf("%d %d\n", temp, result);
     for (int i = 0; i < 9; i++)
         printf("%d ", a[i]);
     printf("%d\n", a[i]);

     result = sum(i, m , n, a[i]);
     printf("%d %d\n", temp, result);
     for (int i = 0; i < 9; i++)
         printf("%d ", a[i]);
     printf("%d\n", a[i]);

     n = 2;
     result = sum(i, m , n, sum(j, m, n, b[i][j]));  // 1
     printf("%d %d\n", temp, result);                // 2
     for (int i = 0; i < 3; i++)                     // 3
         printf("%d ", a[i]);                        // 4
     printf("%d\n", a[i]);                           // 5
}
```

- What is the output of the program if functions are called *by value*? If functions are called by value, the execution is equivalent to the following

```
int temp = 0;

int sum(int i, int m, int n, int ai)
{
    for (i = m; i < n; i++) {
        temp = temp + ai;
    }
    return temp;
}

int main()
{
     int a[9] = {1,10,10,10,100,100,100,100,100};
     int b[4][4] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};

     int temp = 4;
     int m = 0;
     int n = 3;
     int i = 1;
     int j = 1;
     int result;

     result = sum(i, m , n, i)
            = sum(1,0,3,1)
                        i_local = 1
                        m_local = 0
                        n_local = 3
                        ai_local = 1
                        for (i_local = 0; i_local < 3 ; i_local++)
                              temp_global = temp_global+ai_local = temp + 1 // updates global
                                                                            // variable temp
                        return temp_global; // temp_global = 3
     result = 3
     printf("%d %d\n", temp, result);    // prints 4 3
     for (int i = 0; i < 9; i++)
         printf("%d ", a[i]);            // prints 1 10 10 10 100 100 100 100 100
     printf("%d\n", a[i]);               // prints a[1] = 10

     result = sum(i, m , n, a[i])
            = sum(1,0,3,10]
                        i_local = 1
                        m_local = 0
                        n_local = 3
                        ai_local = 10
                        for (i_local = 0; i_local < 3 ; i_local++)
                              temp_global = temp_global+ai_local = temp + 10 // updates global
                                                                             // variable temp
                        return temp_global; // temp_global = 33   starts at 3 and adds 10
                                            // 3 times
     result = 33

     printf("%d %d\n", temp, result);    // prints 4 33
     for (int i = 0; i < 9; i++)
         printf("%d ", a[i]);            // prints 1 10 10 10 100 100 100 100 100
     printf("%d\n", a[i]);               // prints 10          (a[1])

// continued next page
```

```
 n = 2;
 result = sum(i, m , n, sum(j, m, n, b[i][j]))
             i_local = i = 1
             m_local = m = 0
             n_local = n = 2
             ai_local = sum(j,m,n,b[i][j])
                              i_local1 = j = 1
                              m_local1 = m = 0
                              n_local1 = n = 2
                              ai_local1 = b[i][j] = b[1][1] = 5
                              for (i_local1 = 0; i_local1 < 2; i_local1++)
                                    temp_global = temp_global + ai_local1 = temp_global + 5
                              return temp_global = 43      // starts at 33 and adds 5 twice
                        = 43
             for (i_local = 0; i_local < 2; i_local++)
                   temp_global = temp_global + ai_local = temp_global + 43
             return temp_global = 129
         = 129
 printf("%d %d\n", temp, result);    // prints 4 129
 for (int i = 0; i < 3; i++)
     printf("%d ", a[i]);            // prints 1 10 10
 printf("%d\n", a[i]);               // prints 10
}
```

- What is the output of the program if functions are called *by reference*? For this part do not execute lines //1 through //5

```
int temp = 0;

int sum(int i, int m, int n, int ai)
{
    for (i = m; i < n; i++) {
        temp = temp + ai;
    }
    return temp;
}

int main()
{
      int a[9] = {1,10,10,10,100,100,100,100,100};
      int b[4][4] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
      int temp = 4;
      int m = 0;
      int n = 3;
      int i = 1;
      int j = 1;
      int result;

      result = sum(i, m , n, i)
                  i_local     alias of i
                  m_local     alias of m
                  n_local     alias of n
                  ai_local    alias of i

                  // for (i_local = m; i_local < n; i_local++)
                  //    temp_global = temp_global + ai_local
                  //
                  // is equivalent to
                  //
                  for (i = m; i < n; i++)
                        temp_global = temp_global + i;
                  //
                  // at the end of loop
                  // temp_global = 0 (initial) + 0 + 1 + 2 = 3
                  // i = 3
            = 3

      printf("%d %d\n", temp, result);  // print 4 3
      for (int i = 0; i < 9; i++)
            printf("%d ", a[i]);        // print 1 10 10 10 100 100 100 100
      printf("%d\n", a[i]);             // prints 10      (a[1])

      result = sum(i, m , n, a[i]);
                  i_local     alias of i
                  m_local     alias of m
                  n_local     alias of n
                  ai_local    alias of a[i] = a[1]

                  // like above, the loop is equivalent to
                  //
                  for (i = m; i < n; i++)
                        temp_global = temp_global + a[1];
                  //
                  // at the end of loop
                  // temp_global = 3 (initial) + 10 + 10 + 10
                  // i = 3
            = 33

      printf("%d %d\n", temp, result);   // print 4 33
      for (int i = 0; i < 9; i++)
            printf("%d ", a[i]);        // print 1 10 10 10 100 100 100 100
      printf("%d\n", a[i]);             // prints 10      (a[1])
```

- What is the output of the program if functions are called *by name*? The execution is equivalent to the following:

```
int main()
{
     int a[9] = {1,10,10,10,100,100,100,100,100};
     int b[4][4] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};

     int temp1 = 4;
     int m = 0;
     int n = 3;
     int i = 1;
     int j = 1;
     int result;

    // result = sum(i, m , n, i);
        for (i = m; i < n; i++) {
                temp = temp + i;
        }
        result = temp;

     printf("%d %d\n", temp1, result);
     for (int i = 0; i < 9; i++)
         printf("%d ", a[i]);
     printf("%d\n", a[i]);

     //result = sum(i, m , n, a[i]);
        for (i = m; i < n; i++) {
                temp = temp + a[i];
        }
        result = temp;

     printf("%d %d\n", temp1, result);
     for (int i = 0; i < 9; i++)
         printf("%d ", a[i]);
     printf("%d\n", a[i]);

     n = 2;
     // result = sum(i, m , n, sum(j, m, n, b[i][j]));
        for (i = m; i < n; i++) {
               // temp = temp + sum(j, m, n, b[i][j]);
                int t1 = temp;
                int result; //  = sum(j, m, n, b[i][j]);
                        for (j = m; j < n; j++) {
                             temp = temp + b[i][j];
                        }
                    result = temp;
                temp = t1+result;
        }
        result = temp;

     printf("%d %d\n", temp1, result);
     for (int i = 0; i < 3; i++)
         printf("%d ", a[i]);
     printf("%d\n", a[i]);
}
```

The output is

```
4 3
1 10 10 10 100 100 100 100 100 10
4 24
1 10 10 10 100 100 100 100 100 10
4 107
1 10 10 10
```

Problem 2. Consider the following program written in Ada syntax with the execution stack shown on the right side. The line numbers are used to refer to the code and are not part of the code.

```
procedure env is
        x_env: integer;
        y_env: integer;

        procedure a is
                x_a: integer;
                procedure d is
                begin
                        b;
                end d;
        begin
                d;
        end a;

        procedure b is
                x_b: integer;
                procedure c is
                        x_c: integer;
                        procedure e is
                        x_e : integer;
                        begin
                          x_e = x_c + x_env ;
                          x_e = x_b ;
                          env;
                        end;
                begin
                        e;
                end c;
        begin
                c;
        end b;

begin
        a;
end env;
```

```
                 _____________
env   |_____________| 1
 a    |_____________| 2
 d    |_____________| 3
 b    |_____________| 4
 c    |_____________| 5
 e    |_____________| 6
env   |_____________| 7
 a    |_____________| 8
 d    |_____________| 9
 b    |_____________| 10
 c    |_____________| 11
 e    |_____________| 12
```

- Give the low-level code (in terms of mem[]) to setup the access link for activation record 5

  b calls c: b is the defining environment of c, so the access link is equal to the control link

  $mem[sp+AL_{offset}] = fp$

- Give the low-level code (in terms of mem[]) to setup the access link for activation record 6

  c calls e: c is the defining environment of e, so the access link is equal to the control link

  $mem[sp+AL_{offset}] = fp$

- Give the low-level code (in terms of mem[]) to setup the access link for activation record 7

  e calls env: e is three nesting levels deeper than env, so we need to traverse three access links starting from e which takes us to the earlier frame of env, then we copy the access link from that frame

  ```
  temp = fp                          // temp points to frame of e
  temp =  mem[temp+AL_offset]        // temp points to frame of c
  temp =  mem[temp+AL_offset]        // temp points to frame of b
  temp =  mem[temp+AL_offset]        // temp points to frame of env
  temp = mem[temp+ AL_offset]        // temp is access link of env
  mem[sp+ AL_offset] = temp          // copy access link of previous frame of env to new frame of env
                                     // note we are using sp because the code is executed by e
                                     // before the new env is active
  ```

- Give the low-level code (in terms of mem[]) to setup the access link for activation record 8

  env calls a: env is the defining environment of a, so the access link is equal to the control link

  ```
  mem[sp+AL_offset] = fp
  ```

- Give the low-level code (in terms of mem[ ]) for `x_e = x_c + x_env; in procedure e`

  x_e is a local variable:

  ```
  fp+e_offset                        // address of x_e
  ```

  x_c is in c which is one nesting level up:

  ```
  temp1 = mem[fp+AL_offset]          // pointer to frame of c
  temp1+x_c_offset                   // address of x_c
  ```

  x_env is in env, three nesting levels up

  ```
  temp2 = mem[fp+AL_offset]          // pointer to frame of c
  temp2 = mem[temp+AL_offset]        // pointer to frame of b
  temp2 = mem[temp+AL_offset]        // pointer to frame of env
  temp2+x_env_offset                 // address of x_env
  ```

  So, the full code to execute is

  ```
  temp1 = mem[fp+AL_offset]
  temp2 = mem[fp+AL_offset]
  temp2 = mem[temp+AL_offset]
  temp2 = mem[temp+AL_offset]
  mem[fp+e_offset] = mem[temp1+x_c_offset] + mem[temp2+x_env_offset ]
  ```

- Give the low-level code (in terms of mem[ ]) for `x_e = x_b;` in procedure e

  x_e is a local variable:

  fp+e_{offset} // address of x_e

  x_b is in b, two nesting levels up

  temp = mem[fp+AL_{offset}] // pointer to frame of c
  temp = mem[temp+AL_{offset}] // pointer to frame of b
  temp+x_b_{offset} // address of x_env

  So, the full code to execute is

  temp = mem[fp+AL_{offset}]
  temp = mem[temp+AL_{offset}]
  mem[fp+e_{offset}] = mem[temp+x_b_{offset}]

Note on Ada syntax. A procedure declaration has the following form

```
procedure p is

        // declaration of local variables and other
        // procedures nested within p

begin
        // body of p
end p;
```

procedure calls are written without parentheses. So, in the code above, the body of env has a call to procedure a in its body.