Moti Begna

CSCI 5106

Homework 6

Problem 1:

```
void swap(int x, int y) {
  int temp = x;
  x = y;
  y = temp;
}

int main() {
  int i = 0;
  int arr[2];
  arr[0] = 1;
  arr[1] = 1;

  swap(i, arr[i])
  printf("i = %d, j = %d\n", i, j);
  return 0;
}
```

a) Call-by-value

After swap: i = 0, arr[0] = 1, arr[1] = 1

b) Call-by-referance

After swap: i = 1, arr[0] = 0, arr[1] = 1

c) Call-By-value-result

After swap: i = 1, arr[0] = 0, arr[1] = 1

c) Call-by-name

After swap: i = 1, arr[0] = 1, arr[1] = 0

Problem 2:

Procedure P Assignments—x := Q, w :=Q

1. Calls to procedure Q are made within procedure P. The control link in Q then pointes to the beginning of the frame for P, while the access link is determined through nesting calculations (further explained later). These calls complete once some termination condition is met within procedure P, which then completes procedure Q, whose value is returned to variable x, w.

2. The access link of the activation record of procedure P points to procedure M, which is the same procedure that the control link points to. This, however, is true only in the first invocation of P. Each further invocation of P as a result of recursion has corresponding control links to procedure that made the recursive call—procedure R. (i.e., during recursion, P_2's control link points to R_1, P_3 to R_2, etc.). The access links, however, will always point back to procedure M. Thus, the variable x in the assignment corresponds to the variable x in the cell allocated by procedure M. The variable w is defined within procedure P, thus new cells for this variable will be allocated during each invocation of P. Once procedure Q is called, however, it's activation record now contains a new control link which simply points to the activation record of its caller (i.e. procedure P), while its access link is determined by—in one implementation—calculating the difference in the nesting depths of the caller and itself, and using this to chain back through access links to one more than this difference. Once procedure Q completes in both instances, the activation records that proceeded the calls are popped from the stack and control is returned back to P. Any returned values are subsequently stored in variables x and w.

3. The compiler initially needs to allocate space within the activation record (which resides on the stack) for the variables defined within the procedure for that activation record. The address to

which control will return from a call to Q will also need to be saved in order to resume the execution of P.

Procedure Q Assignment—y := R

1. The call to procedure R is made within procedure Q. The control link in R then points to the beginning of the frame for Q, while the access link is determined through nesting calculations. This call completes once some termination condition is met within procedure R, whose value is returned to variable y.

2. The access link of the activation record of procedure Q points to procedure P, which is the same procedure that the control link points to. Each further invocation of Q as a result of recursion has corresponding links to P (i.e. Q_1's access/control links point to P_1, Q_2 to P_2, etc.) Thus, the variable y in the assignment within Q_1 corresponds to the variable x in the cell allocated by procedure P_1, while any other instances of this assignment due to recursion would correspond to the variable x within the procedure pointed by the access link. Once procedure R is called, however, it's activation record now contains a new control link which simply points to the activation record of its caller (i.e. procedure Q), while its access link is determined by nesting calculations. Here, R_1's access link will always point to Q_1, R_2 to Q_2, etc. Once procedure R completes, the activation record that proceeded the call is popped from the stack and control is returned back to Q. Any returned values are subsequently stored in variable y.

3. The compiler initially needs to allocate space within the activation record (which resides on the stack) for the variables defined within the procedure for that activation record. The address to

which control will return from a call to R will also need to be saved in order to resume the execution of Q.

Procedure R Assignment—z := P

1. The recursive call to procedure P_n is made within procedure R_{n-1}. The control link in P_n then points to the beginning of the frame for R_{n-1}, while the access link is determined through nesting calculations. This call completes once some termination condition is met within procedure P_n, whose value is returned to variable z.

2. The access link of the activation record of procedure R points to procedure Q, which is the same procedure that the control link points to. Each further invocation of R as a result of recursion has corresponding links to Q (i.e. R_1's access/control links point to Q_1, R_2 to Q_2, etc.) Thus, the variable z in the assignment within R_1 corresponds to the variable z in the cell that exists within the scope of Q_1, which was thus defined within P_1, while any other instances of this assignment due to recursion would correspond to the variable z within the procedure pointed by that access link. Once procedure P is called, however, it's activation record now contains a new control link which simply points to the activation record of its caller (i.e. procedure R), while its access link is determined by nesting calculations. Here, since a recursive call is being made, the access link would always point to procedure M. Once procedure P completes, the activation record that proceeded the call is popped from the stack and control is returned back to R. Any returned values are subsequently stored in variable z.

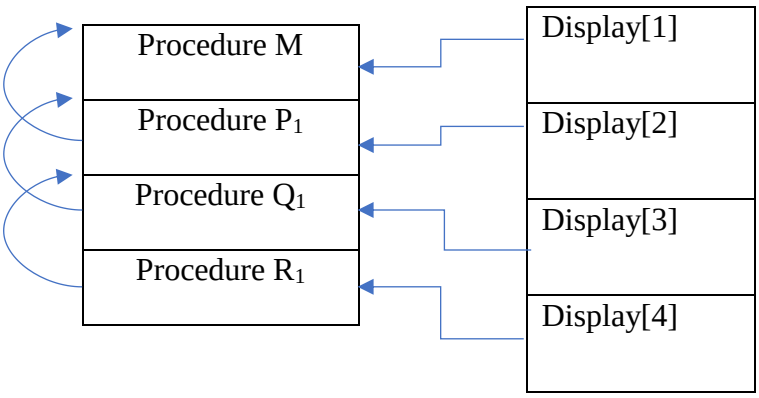
3. The compiler initially needs to allocate space within the activation record (which resides on the stack) for the variables defined within the procedure for that activation record. The address to which control will return from a call to P will also need to be saved in order to resume the

execution of R. The compiler, however, will not be able to statically allocate space in each recursive call the number of invocations is a priori unknown.

Problem 3:

(Discussion on general affects of using displays vs. access links)

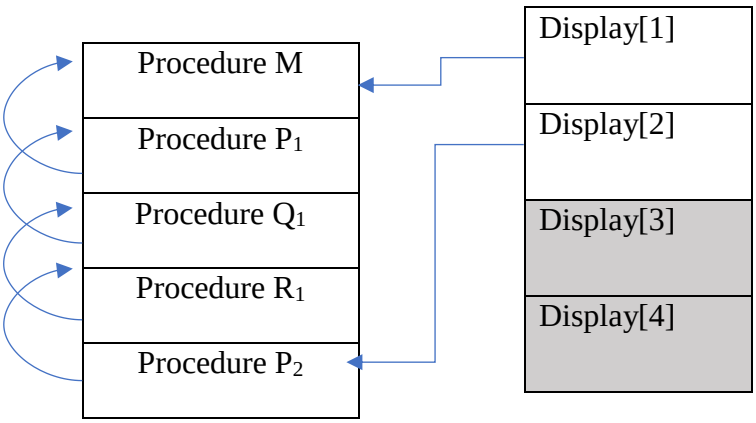
When using displays, each i^{th} entry of the display points to the activation record of the procedure at the i^{th} level of the program. Thus, an array of for one loop of the program (right before the recursive call) would look as follows:



Here, once procedure's P_1, Q_1, and R_1 are called, the display is set up such that each procedure will have access to the procedures in the display that are visible to it. Thus, when P_1 is initially called, it only has access to the display element pointing to the activation record of M, while when R_1 is called, it will have the activation record for M, P_1, and Q_1. The control links will always point to the caller of a procedure, which is no different than the previous implementation. Once a procedure completes, however, the display would be restored to the state it was in prior to the call to that procedure.

During a recursive call, such as the case when P is once again called in R, the display would end up changing so that the new invocation of the procedure will only have access to the activation

records that exist within its lexical scope. In this case, a call to P_2 would make it so that only Display[1] and Display[2] are accessible.



In order to find the correct values of variables w, x, y, and z, a procedure would simply calculate their location as a displacement from a predetermined position Display[i].