Homework 5

COMP 3220

Due: October 15, Wednesday by 11:59AM

Please submit as a PDF or WORD document using Canvas

**(1. 15pts) Question 9.5 (page 438) from the textbook.**

Consider the following program written in C syntax:

**void** swap(**int** a, **int** b) {

**int** temp;

temp = a;

a = b;

b = temp;

}

**void** main() {

**int** value = 2, list[5] = {1, 3, 5, 7, 9};

swap(value, list[0]);

swap(list[0], list[1]);

swap(value, list[value]);

}

For each of the following parameter-passing methods, what are all of the values of the variables value and list after each of the three calls to swap?

\*Note: Answers are after each swap:

swap(value, list[0]);

swap(list[0], list[1]);

swap(value, list[value]);

- a. Passed by value

Value = 2, List = { 1, 3, 5, 7, 9 }

Value = 2, List = { 1, 3, 5, 7, 9 }

Value = 2, List = { 1, 3, 5, 7, 9 }

- b. Passed by reference

Value = 1, List = { 2, 3, 5, 7, 9 }

Value = 1, List = { 3, 2, 5, 7, 9 }

Value = 2, List = { 3, 1, 5, 7, 9 }

- c. Passed by value-result

Value = 1, List = { 2, 3, 5, 7, 9 }

Value = 1, List = { 3, 2, 5, 7, 9 }

Value = 2, List = { 3, 1, 5, 7, 9 }

**(2. 15pts) Question 9.7 (page 438) from the textbook.**

Consider the following program written in C syntax:

**void** fun (**int** first, **int** second) {

first += first;

Programming Exercises **439**

second += second;

}

**void** main() {

**int** list[2] = {1, 3};

fun(list[0], list[1]);

}

For each of the following parameter-passing methods, what are the values of the list array after execution?

- a. Passed by value

List = {1, 3}

- b. Passed by reference

List = {2, 6}

- c. Passed by value-result

List = {2, 6}

**(3. 15pts) Question 9.12 (page 439) from the textbook.**

Research Jensen’s Device, which was a widely known use of pass-by-name parameters, and write a short description of what it is and how it can be used.

- Jensen’s Device exploits pass-by-name. Call by name is an argument passing convention that delays the evaluation of an argument until it is actually used in the procedure (a consequence of the copy rule for procedures). Algol introduced call by name.

The body of a function is interpreted at call time after textually substituting the actual parameters into the function body. In this sense the evaluation method is similar to that of C preprocessor macros. By substituting the actual parameters into the function body, the function body can both read and write the given parameters. In this sense the evaluation method is similar to pass-by-reference. The difference is that since with pass-by-name the parameter is evaluated inside the function, a parameter such as a[i] depends on the current value of i inside the function, rather than referring to the value at a[i] before the function was called.

Advantages:

* It has a simple semantic model as textual substitution.
* Modification and re-evaluation of argument expressions has useful applications, such as Jensen's device.
* Argument expressions are not necessarily evaluated:

boolean procedure and (x, y);

boolean x, y;

begin

if x then return y else return false

end;

Here, y is not evaluated if x is false.

**(4. 15pts) Question 9.15 (page 439) from the textbook.**

What are at least two arguments against the use of pass-by-name parameters?

- Disadvantages:

* Repeated evaluation of arguments can be inefficient.
* It can have unsafe semantic effects.
* It is difficult to implement. Argument expressions must be compiled to special parameter-less procedures called thunks. These thunks are passed into the called procedure and used whenever necessary to evaluate or re-evaluate the argument.

**(5. 15pts) Question 10.1 (page 467) from the textbook.**

Show the stack with all activation record instances, including static and dynamic chains, when execution reaches position 1 in the following skeletal program. Assume Bigsub is at level 1.

**procedure** Bigsub **is**

**procedure** A **is**

**procedure** B **is**

**begin** -- of B

. . .

**end**; -- of B

**procedure** C **is**

**begin** -- of C

. . .

B;

. . .

**end**; -- of C

**begin** -- of A

. . .

C;

. . .

**end**; -- of A

**begin** -- of Bigsub

. . .

A;

. . .

**end**; -- of Bigsub

{

|  |
| --- |
| Dynamic Link |
| Static Link |
| Return Address to C |
| Dynamic Link |
| Static Link |
| Return Address to A |
| Dynamic Link |
| Static Link |
| Return Address to Bigsub |
| Dynamic Link |
| Static Link |
| Return address to caller |
| . |
| . |
| stack |

Activation Record Instance for B

ARI for Bigsub

ARI for C

ARI for A

{

{

{

**(6. 15pts) Question 10.3 (page 468-469) from the textbook.**

Show the stack with all activation record instances, including static and dynamic chains, when execution reaches position 1 in the following skeletal program. Assume Bigsub is at level 1.

**procedure** Bigsub **is**

**procedure** A(Flag : Boolean) **is**

**procedure** B **is**

**begin** -- of B

. . .

A(false);

**end**; -- of B

**begin** -- of A

**if** flag

**then** B;

**else** C;

. . .

**end**; -- of A

**procedure** C **is**

**procedure** D **is**

**begin**-- of A

. . .

**end**; -- of D

. . .

D;

**end**; -- of C

**begin** -- of Bigsub

. . .

A(**true**);

. . .

**end**; -- of Bigsub

The calling sequence for this program for execution to reach D is

Bigsub calls A

A calls B

B calls A

A calls C

C calls D

{

|  |
| --- |
| Dynamic Link |
| Static Link |
| Return Address to C |
| Dynamic Link |
| Static Link |
| Return Address to A |
| Parameter (Flag) |
| Dynamic Link |
| Static Link |
| Return Address to B |
| Dynamic Link |
| Static Link |
| Return Address to A |
| Parameter (Flag) |
| Dynamic Link |
| Static Link |
| Return Address to Bigsub |
| Dynamic Link |
| Static Link |
| Return address to caller |
| . |
| . |
| stack |

ARI for D

{

ARI for Bigsub

ARI for B

ARI for A

{

{

{

{

ARI for A

ARI for C

**(7. 10pts) Question 10.10 (page 470) from the textbook.**

Design a skeletal program and a calling sequence that results in an activation record instance in which the static and dynamic links point to different activation-recorded instances in the run-time stack.

**procedure** main **is**

X: Integer;

**procedure** Double **is**

**begin** -- of Double

Work;

**end**; -- of Double

**procedure** Work **is**

**begin** -- of Work

X:= X \* 2;

**end**; -- of Work

**begin** -- of main

X: = 2

Double;

**end**; -- of main

|  |
| --- |
| Local Variable |
| Dynamic Link to Double |
| Static Link to main |
| Return Address to Double |
| Dynamic Link to main |
| Static Link to main |
| Return address to main |
| Local Variable |

{

ARI for Work

ARI for Double

ARI for main

{

{