COMP.2030 Lab 1 9/13/22

NAME: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Pseudo-C is a subset of C that limits the use of control-flow statements, like while loops. Instead, the only control statements allowed in pseudo-C are:
   1. Labels (with any name of your choice) and goto label\_name
   2. An unconditional jump of the form goto label\_name
   3. A conditional jump of the form if (condition) goto label\_name

Rewrite the C function below into pseudo-C:

int test(int x, int y){

int val = 4\*x;

if (y > 0) {

if (x < y) {

val = x-y;

} else {

val = x^y;

}

} else {

if (y < -2) {

val = x+y;

}

}

return val;

}

There are two main variants, depending on if you put the “then” clause first, or the “else” branch first in the pseudo-C code. Mixing and matching is fine, as long as the correct choice of the three options for reassigning ‘val’ are taken under the same conditions as the left.

// ‘else’ first

int test(int x, int y){

int val = 4\*x;

if (y > 0) goto option1or2;

if (y < -2) goto option3;

goto join;

option3:

val = x+y;

goto join;

option1or2:

if (x < y) goto option1;

val = x^y;

goto join;

option1:

val = x-y;

join:

return val;

}

// ‘then’ first

int test(int x, int y){

int val = 4\*x;

if (y <= 0) goto option3;

if (x >= y) goto option2;

val = x-y;

goto join;

option2:

val = x^y;

goto join;

option3:

if (y >= -2) goto join;

val = x+y;

join:

return val;

}

1. Suppose you have an integer array A with N elements (A and N are global). The array A is sorted in ascending order. Function bin\_search() performs a binary search of A looking for a key and returns the index of the key (or -1 if not found). The following C code on the left continuously subdivides a range in halves until the key is found or min and max cross each other. In the space on the right, convert this C code to a Pseudo-C version:

int bin\_search(int key) {

int found = 0;

int j = 1;

int min = 0;

int max = N-1;

loop:

if (found != 0) goto done;

if (min > max) goto done;

int sum = min + max;

int half = sum / 2;

j = floor(half);

int elem = A[j];

if (key != elem) goto not\_found;

found = 1;

goto join;

not\_found:

if (key > elem) goto bigger;

max = j – 1;

goto join;

bigger:

min = j + 1;

join:

goto loop;

done:

return j;

}

int bin\_search(int key) {

int found = 0;

int j = 1;

int min = 0;

int max = N-1;

while (found != 0) {

if (min > max) {

break;

} else {

j=floor((min+max)/2);

if (key == A[j]) {

found = 1;

} else {

if (key < A[j]) {

max = j - 1;

} else {

min = j + 1;

}

}

}

return j;

}

This answer has many correct variations, depending on how the student optimizes the code or swaps the conditions for the jumps. Some more example correct answers:

int bin\_search(int key) {

int j = 1;

int min = 0;

int max = N-1;

loop:

if (min > max) goto done;

int sum = min + max;

int half = sum / 2;

j = floor((min+max)/2);

int elem = A[j];

if (key != elem) goto done;

not\_found:

if (key > elem) goto bigger;

max = j – 1;

goto loop;

bigger:

min = j + 1;

goto loop;

done:

return j;

}

int bin\_search(int key) {

int found = 0;

int j = 1;

int min = 0;

int max = N-1;

loop:

if (found != 0) goto done;

if (min > max) goto done;

j = floor((min+max)/2);

if (key != A[j]) goto not\_found;

found = 1;

goto join;

not\_found:

if (key > A[j]) goto bigger;

max = j – 1;

goto join;

bigger:

min = j + 1;

join:

goto loop;

done:

return j;

}

int bin\_search(int key) {

int found = 0;

int j = 1;

int min = 0;

int max = N-1;

loop:

if (found != 0) goto done;

if (min > max) goto done;

j = floor((min+max)/2);

if (key == A[j]) goto same;

if (key < A[j]) goto smaller;

min = j + 1;

goto join;

smaller:

max = j – 1;

goto join;

same:

found = 1;

join:

goto loop;

done:

return j;

}