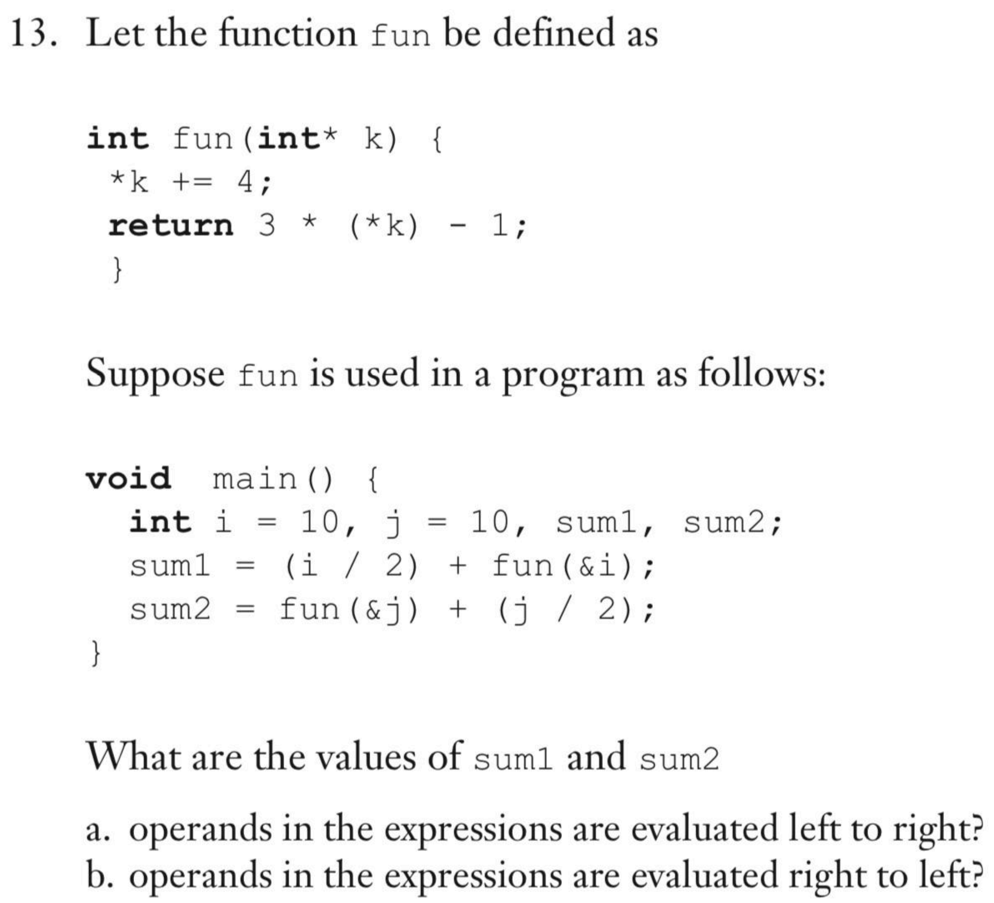
Jeongbin Son
CS403
2/7/24

Problem 1)



13. Let the function `fun` be defined as

```
int fun(int* k) {
 *k += 4;
 return 3 * (*k) - 1;
 }
```

Suppose `fun` is used in a program as follows:

```
void  main() {
  int i = 10, j = 10, sum1, sum2;
  sum1 = (i / 2) + fun(&i);
  sum2 = fun(&j) + (j / 2);
}
```

What are the values of `sum1` and `sum2`

a. operands in the expressions are evaluated left to right?
b. operands in the expressions are evaluated right to left?

---

a.

i = 10;
sum1 = (10 / 2) + fun(10);
fun(10), i = 10 + 4, return 3 * (14) - 1 = 41
sum1 = 5 + 41 = 46

sum2 = fun(10) + (j / 2)
fun(10), j = 10 + 4, return 3 * (14) - 1 = 41

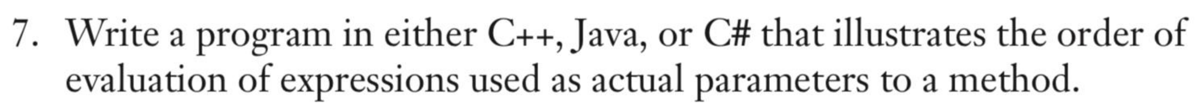
j = 14
sum2 = 41 + (14/2) = 48

b.

i = 10
sum1 = fun(&i) + (i / 2)
fun(&i), i = 10 + 4, return 3 * (14) - 1 = 41
i / 2, where i = 14, 14 / 2 = 7
sum1 = 41 + 7 = 48

sum2 = (j / 2) + fun(&j)
sum2 = (10 / 2) + fun(&j)
fun(&j), j = 10 + 4, return 3 * (14) - 1 = 41
sum2 = 5 + 41 = 46

Problem 2)

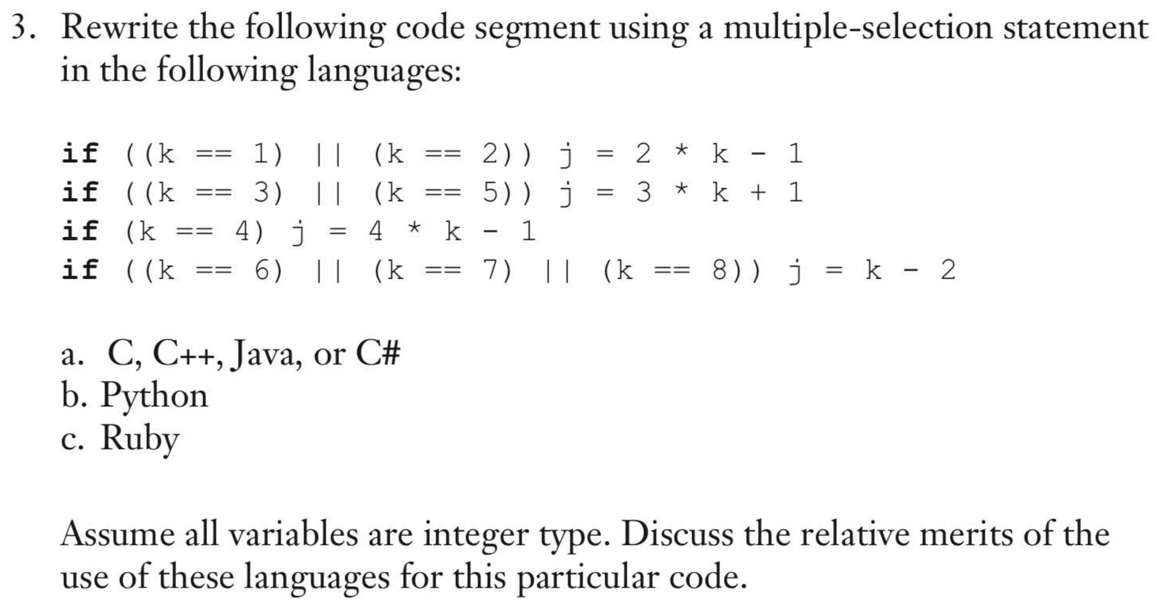


7. Write a program in either C++, Java, or C# that illustrates the order of evaluation of expressions used as actual parameters to a method.

```
#include <iostream>
using namespace std;

int compute(int n) {
   cout << "Evaluating " << n << endl;
   return n;
}

void display(int a, int b, int c) {
   cout << "Inside display function: " << a << ", " << b << ", " << c << endl;
}

int main() {
   cout << "Calling display function" << endl;
   display(compute(1), compute(2), compute(3));
```

```
    return 0;
}
```

Problem 3)



3. Rewrite the following code segment using a multiple-selection statement in the following languages:

```
if ((k == 1) || (k == 2)) j = 2 * k - 1
if ((k == 3) || (k == 5)) j = 3 * k + 1
if (k == 4) j = 4 * k - 1
if ((k == 6) || (k == 7) || (k == 8)) j = k - 2
```

a. C, C++, Java, or C#
b. Python
c. Ruby

Assume all variables are integer type. Discuss the relative merits of the use of these languages for this particular code.

C++

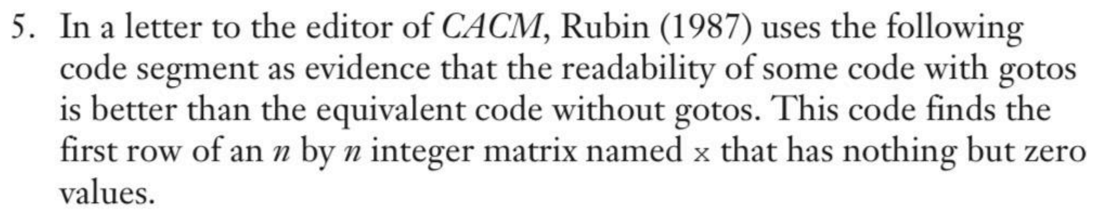
```
int j;
switch (k) {
    case 1:
    case 2:
        j = 2 * k - 1;
        break;
    case 3:
    case 5:
        j = 3 * k + 1;
        break;
    case 4:
        j = 4 * k - 1;
        break;
    case 6:
    case 7:
    case 8:
        j = k - 2;
        break;
    // default case if needed
}
```

Python
```
if k == 1 or k == 2:
    j = 2 * k - 1
elif k == 3 or k == 5:
    j = 3 * k + 1
elif k == 4:
    j = 4 * k - 1
elif k == 6 or k == 7 or k == 8:
    j = k - 2
else:
    # handle case if k does not match any of the specified values
    # j = default_value or any other operation
```

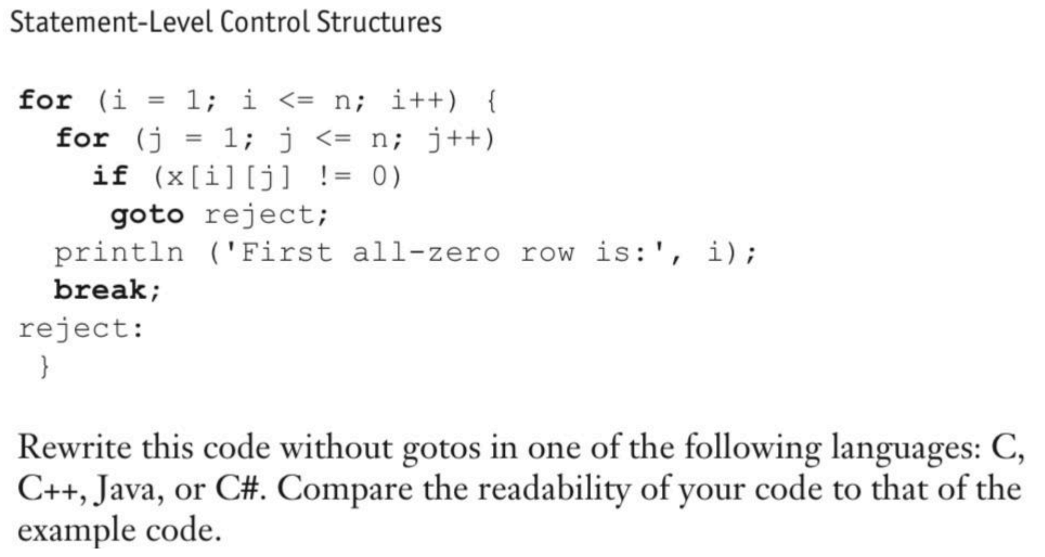
Ruby
```
case k
when 1, 2
    j = 2 * k - 1
when 3, 5
    j = 3 * k + 1
when 4
    j = 4 * k - 1
when 6, 7, 8
    j = k - 2
else
    # handle default case if necessary
end
```

C++: Ideal for performance-critical applications due to its efficiency and control over system resources, but involves more complexity and longer development time.
Python: Offers rapid development and ease of use with readable syntax, making it perfect for quick prototyping and applications where development speed is crucial, though it may have slower execution compared to C++.
Ruby: Known for its elegant and concise syntax, Ruby enhances productivity and is great for web development, but like Python, it may not match C++ in terms of execution speed for resource-intensive tasks.

Problem 4)



5. In a letter to the editor of *CACM*, Rubin (1987) uses the following code segment as evidence that the readability of some code with gotos is better than the equivalent code without gotos. This code finds the first row of an *n* by *n* integer matrix named x that has nothing but zero values.



```
for (i = 1; i <= n; i++) {
  for (j = 1; j <= n; j++)
    if (x[i][j] != 0)
     goto reject;
  println ('First all-zero row is:', i);
  break;
reject:
 }
```

Rewrite this code without gotos in one of the following languages: C, C++, Java, or C#. Compare the readability of your code to that of the example code.

```
for (i = 1; i <= n; i++) {
        for (j = 1; j <= n; j++)
                if (x[i][j] != 0)
                        rejectFunction();
                println ('First all-zero row is:', i);
                break;

}
```

```
void rejectFunction() {
        //code to reject
}
```

OR

```
#include <iostream>
using namespace std;

int main() {
    int n = /* matrix size */;

    int x[n][n] = {/* initialize the matrix with values */};
    bool found = false;

    for (int i = 0; i < n; i++) {
        bool allZero = true;
        for (int j = 0; j < n; j++) {
            if (x[i][j] != 0) {
                allZero = false;
                break; // Exit the inner loop if a non-zero is found
            }
        }
        if (allZero) {
            cout << "First all-zero row is: " << i << endl;
            found = true;
            break; // Exit the outer loop if an all-zero row is found
        }
    }

    if (!found) {
        cout << "No all-zero row found." << endl;
    }

    return 0;
}
```

The use of `goto` can be considered better in this specific context for its efficiency in breaking out of nested loops without additional variables or logic. There’s no need to access another function to review the code from a readability perspective.