C Programming

Noureddine Hamid - - Redone Mahjoubi

TP 02

Exercice 1:

Write a function template named **multiply** that allows the user to multiply one value of any type (first parameter) and an integer (second parameter). The function should return the same type as the first parameter.
The following program should run:

```
#include <iostream>

// write your multiply function template here

int main()
{
        std::cout << multiply(2, 3) << '\n';
        std::cout << multiply(1.2, 3) << '\n';

        return 0;
}
```

Output:
6
3.6

Exercice 2:

Binary exponentiation (also known as exponentiation by squaring) is an algorithm which allows to calculate a^n using only $O(\log n)$ multiplications instead of $O(n)$ required by the standard approach.

Write a function template named **powiter** that allows the user to calculate a^n, where the first parameter a could be of any type and the second parameter n is an integer. The function should be not be recursive and should return the same type as the first parameter.

Write a function template **powrec** which does the same thing in a recursive way.

```cpp
#include <iostream>

// write your powiter function template here
// write your powrec function template here

int main()
{
        std::cout << powiter(2, 3) << '\n';
        std::cout << powiter(1.2, 3) << '\n';

        std::cout << powrec(2, 3) << '\n';
        std::cout << powrec(1.2, 3) << '\n';

        return 0;
}
```

Output:

8
1.728
8
1.728

Exercice 3:

Write a function template named **matpow** that allows the user to calculate a^k, where the first parameter a is an $n * n$ matrix of any type (integer or float) and the second parameter k is an integer. The function should return the matrix a^k.

Write a function template named **matprint** that prints the matrix returned by **matpow**

Write a function template named **matpownaive** that behaves similarly to the **matpow** but calculate a^k in the naive way (multiplying a by itself k times). Compare the time complexity of **matpow** and **matpownaive** for big values of k.

```cpp
#include <iostream>
#include <vector>

// write your matpow function template here
// write your matprint function template here
// write your matpownaive function template here

int main (int argc, char *argv[])
{
    std::vector<std::vector<int> > a { { 1, 1, 1 },
                                       { 2, 2, 2 } };

    std::vector<std::vector<int> > b { { 1.0, 1.23, 1.22 },
```

```
                                        { 2.22, 2.33, 2.33 } };
    matprint(matpow(a, 12));
    matprint(matpow(b, 15));
}
```