INDIAN ASSOCIATION FOR THE CULTIVATION OF SCIENCE

SCHOOL OF MATHEMATICAL & COMPUTATIONAL SCIENCES

PG - I (SMCS), Autumn Semester 2022-23

Instructor: **Debarshi Kumar Sanyal**
Course: **COM 4111: Object-Oriented Programming with C++**

Assignment Number: **4**
Date: **Sep 21, 2022**

INSTRUCTIONS

(1) Create a folder with the name `Assignment_4_$name$` (where `$name$` denotes name of the student). In that folder create sub-folders for each problem (example: `prob_0, prob_1, prob_2`, etc.). Write your codes for a particular problem in the respective folder (e.g., `prob1` might contain two C++ files `swap.cpp` and `swap2.cpp`)

(2) Give meaningful comments to explain the method used in your program. Programs without valid comments will be evaluated to zero marks.

Problem 1 Declare a class to model a date. Apart from the day, month, and year, it should store a string representation of the date, initialized to an empty string. The class should have suitable constructor(s), *get* methods, and *set* methods. The first time a user requests for the string representation of the date, it should be generated and stored in a cache in the object before returning it to the user, so that subsequent requests can be satisfied from the cache (until the date changes). The class should have member functions to (1) check if the date belongs to a leap year, (2) add years to the date and form a new date. You may add other useful member functions. Use `const` and `mutable` at appropriate places. [`date.cpp`]

Problem 2 Declare a class to model a complex number. The class should have member functions to support addition, subtraction, multiplication and division of complex numbers. Note that the user should be able to use these functions to multiply a complex number with a real number or another complex number. Write a program showing a few use cases of the class. [complex.cpp]

Problem 3 Write a C++ program to input the length of a string, dynamically allocate memory for the string, accept the string from the user, check and report if the string is a palindrome (i.e., reads the same forward and backward), and print the reversed string if it is not a palindrome. [palindrome.cpp]

Problem 4 Suppose you are required to implement a linear search algorithm in C++ which can operate on arrays of different types. The function declaration looks approximately like the following: `int linear_search(void* p_target, void* arr, int N, comp pf);` Here, `pf` denotes a pointer to a function that can compare two elements of the same type and return a value denoting whether they are *equal* or not. Function `linear_search` returns the index of the array `arr` (containing `N` elements) at which the target element (pointed to by `p_target`) is present , or -1 if the target element is absent in the array.
The function can be used as follows:

```
int main() {
    //...
    int arrInt[] = {1,5,-2,20,6};
    int targetInt = 15;
    void* p_target = &targetInt;
    int N = sizeof(arrInt)/sizeof(int);
    int index = linear_search(p_target, arrInt, N, pfInt); //pfInt ?
    //...
    double arrDouble[] = {2.3,4.6,-1.2};
    double targetDouble = 2.3;
    p_target = &targetDouble;
    N = sizeof(arrDouble)/sizeof(double);
    index = linear_search(p_target, arrDouble, N, pfDouble); //pfDouble ?
    //...
    return 0;
}
```

Write a correct definition for the function `linear_search` and use suitable comparator functions so that the functionality indicated in the `main()` function can be achieved. [versatile_search.cpp]

Problem 5 According to `https://en.cppreference.com/w/cpp/language/overload_resolution`, the rules for C++ function overload resolution are: "For each pair of viable function F1 and F2, F1 is determined to be a better function than F2 if implicit conversions for all arguments of F1 are *not* worse than the implicit conversions for all arguments of F2, and there is at least one argument of F1 whose implicit conversion is better than the corresponding implicit conversion for that argument of F2, ... These pair-wise comparisons are applied to all viable functions. If exactly one viable function

is better than all others, overload resolution succeeds and this function is called. Otherwise, compilation fails.".
According to ISO C++ standards ISO/IEC 14882:2020 6th edition (`https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2020/n4849.pdf`), function overload resolution is done as follows: "First, a subset of the candidate functions (those that have the proper number of arguments and meet certain other conditions) is selected to form a set of viable functions. Then the best viable function is selected based on the implicit conversion sequences needed to match each argument to the corresponding parameter of each viable function. If a best viable function exists and is unique, overload resolution succeeds and produces it as the result. Otherwise overload resolution fails and the invocation is ill-formed."
Verify the above rules for the following cases.

1.
```
#include <iostream>
void f(char, short, double) { std::cout<<"f1"<<std::endl; }
void f(int, int, int) {std::cout<<"f2"<<std::endl; }

void F(int x, int y) {std::cout<<"F1 "<<std::endl;    }
void F(char x, double y) { std::cout << "F2" << std::endl; }

void k(double, double) {std::cout<<"k1"<<std::endl;}
void k(int, int) {std::cout<<"k2"<<std::endl;}

int main() {
    f('a', (short)2, (short)3);
    f('a',3,2);
    f('a',3,2.2);
    F(2,3);
    F('a',5.4);
    F('a', 5);
    F('a','b');
    F(2,2.5);
    k(1.2,2.1);
    k(1,2);
    k(2.3,2);
    k(long(0), bool(0));
    return 0;
}
```

2.
```
#include <iostream>
void f(int) { std::cout<<"f(int)"<<std::endl; }
void f(int =0) {std::cout<<"f(int =0)"<<std::endl;}
int main() {
    f(5);
    f(0);
    f();
    return 0;
}
```