ID: …………………………………………………………….

Name: …………………………………………………….

IT603 – Programming

FINAL EXAM – Autumn 2019-20

Max. Marks: 40 Time: 2 hours

Instructions:

1. *Q1 to Q8 carry 1 mark each. Circle the letter against your choice of answer for these questions.*
2. *Rest of the questions have their marks written alongside them*
3. *No queries / clarifications are allowed*
4. *Use of any other construct in place of STL will not get marks. Eg. Using char* or char[] instead of std::string is deemed incorrect.*
5. *All answers to be written on question paper only at space provided for the same*

Q1. Which of the following is not an STL adaptive container?

a) stack
b) vector
c) queue
d) priority_queue

Q2. Which of the following is correct syntax to create a vector v of 10 integers and initialize all elements of it to 11

a) vector<int> v (10,11)
b) vector<int> v {10,11}
c) vector<int> v (11,10)
d) vector<int> v {11,10}

Q3. The STL container that allows efficient insertions to both ends of the sequence is

a) queue
b) priority_queue
c) deque
d) vector

Q4. When a template function is defined in a program, that function is ‘seen’ by

a) compiler
b) linker
c) compiler, linker
d) compiler, linker, OS

Q5. For the following definition of template function

```
template <typename T> T larger(T a, T b)
{
	return a>b ? a : b;
}
```

how would you call this function passing it two string params a and b?

a) template<string> larger(string a, string b);
b) template<string>larger(a, b);
c) larger<string>(a, b);
d) <string>larger(a, b);

Q6. For the template function defined in Q5, you specialize it for a type named Player. The prototype of this specialized template function would be

a) template<> Player larger(Player a, Player b);
b) template<Player> Player larger(Player a, Player b);
c) template<typename T> Player larger(Player a, Player b);
d) template<class Player> Player larger(Player a, Player b);

Q7. How would you declare an iterator itr to a vector of elements where each element is a pair of string and integer?

a) iterator<vector<pair<string, int>::itr;
b) vector<pair<string, int>>::iterator itr;
c) auto itr;
d) auto::iterator<pair<string, int> itr;

Q8. What is the output of following program?

```
void main()
{
	string s{ "abcdef" };
	auto a = s.substr(1, 4);
	s[3] = 'z';
	a[0] = 'x';
	cout << a << ' ' << s << endl;
}
```

a) bcde abcdez
b) xcde abczef
c) xcdz xbczef
d) abcd xbczef

Q9. Someone wrote the following program. You observed that it can be improved in terms of avoiding unnecessary copies of the string objects. So, you commented the PrintSubString() function out:

```
/*
void PrintSubString(string fullString, int startIndex, int count)
{
	if (fullString.length() >= startIndex + count) {
		cout << fullString.substr(startIndex, count);
	}
}
*/
void main()
{
	string s = "This is a long string";
	PrintSubString(s, 4, 10);
}
```

Rewrite the PringSubString() function that you as a programmer would write as an improvement to avoid copies of strings.

(2)

For Q10 to Q14, we'll use following partial program and you will be asked to complete / write the rest of the program based on the questions.

```
enum Gender {
	female, male
};
struct Student {
	string name;	long id;   float cpi;   Gender gender;
};

void PrintStudentInfo(Student s)
{
	cout << s.id << ' ' << s.cpi << ' ' << s.name << ' ' << (s.gender == female ? "F" : "M") << endl;
}

vector<Student> CreateSomeStudents()
{
	vector<Student> v;
	v.push_back({ "Shiraj Govil",201612011, 8.6, male });
	v.push_back({ "Ashita Patel",201612021, 9.1, female });
	v.push_back({ "Pusha Murthy",201612041, 7.6, female });
	v.push_back({ "Bibek Sen",201612001, 8.2, male });
	v.push_back({ "Surit Saren",201612051, 8.6, male });
	return v;
}
```

Q10. Create a struct functor CompareByCpi which compares cpi of two students and returns true if student in first parameter has cpi LESS than student passed in second parameter

(2)

Q11. Assuming struct CompareByCpi functor is available from Q10 above, complete the SortByCpi function below which uses priority queue container and the CompareByCpi functor to print the list of students in vector v sorted by their cpi. Use PrintStudentInfo() to print the student's info.

(2)

```
void SortByCpi()
{
	auto v = CreateSomeStudents();











}
```

Q12. Complete the following function which iterates over the vector v to create another sequence in which all female students are in the front of the queue and male students are placed after them efficiently. Iterate over this new sequence and print the info using PrintStudentInfo().

(2)

```
void FM()
{
	auto v = CreateSomeStudents();











}
```

Q13. Complete the GroupByGender function below. It should use map and vector to group the students into two based on the gender.

(3)

```
void GroupByGender
{
	auto v = CreateSomeStudents();
	vector<Student> males, females;

	………………………………………………………………………………… groups;		// create map named groups

	…………………………………………………………………………………  // assign females vector to female group

	…………………………………………………………………………………  // assign males vector to male group

	for (……………………………………) { //iterate over v using range based for loop
		// based on gender put the student into appropriate group



	}
	cout << "*** Female Students ***" << endl;

	for (auto itr = ……………………………………………………;  itr != ……………………………………………; ++itr)
		PrintStudentInfo(*itr);
```

```
        cout << "*** Male Students ***" << endl;

        for (auto itr = ……………………………………………………….; itr != ……………………………………………….; ++itr)
                PrintStudentInfo(*itr);
}
```

Q14. Complete the function CpiCount below. The idea is to efficiently know how many students have a particular cpi (like, 8.6 in the code below) and print the student's info of all such students. Use appropriate STL container. (3)

```
void CpiCount()
{
        auto v = CreateSomeStudents();

        ……………………………………………………………………………………… cpis;

        for (…………………………………………………) {

                ………………………………………………….
        }

        // create "empty" Student object but with the CPI we want to find
        Student checkCpi = { "", 0, 8.6 };

        int count = cpis………………………………………….(checkCpi); // write the function call

        cout << "Students with CPI " << checkCpi.cpi << " = " << count << endl;

        auto p = cpis……………………………………...(checkCpi);

        for (auto itr = ………………………………….; itr != ……………………………………..; ++itr) {

                PrintStudentInfo(*itr);
        }
}
```

Q15 to Q20 are based on following program description. The program simulates a software library of books. Each book record has title, author(s), keywords and cost.

```
struct Book {
        string title;
        string authors;
        string keywords;
        unsigned int cost; // in INR
};
```

A user can search by author or keyword. If user uses a keyword to search for a book, all books that have that keyword in their keywords string should show in search result. Ditto for search by author.

At launch, the app reads Book records from (simulated) database. Then, prepares for quick search by author's name and by keywords. It also builds a quick data structure to return the cost of the book, given a title.
Here is the main() function. You are required to complete the functions used in main. Follow the comments.

```
void main()
{
        // read from DB
        auto allBooks = ReadFromDB();

        // print all books
        ListAllBooks(allBooks);

        // Prepare Search by authors
        auto byAuthor = IndexBooksToSearchByAuthor(allBooks);

        // Prepare Search by keyword
        auto byKeyword = IndexBooksToSearchByKeyword(allBooks);

        // Prepare Quick way to find cost, given a title
        auto titleToCost = BuildTableToFindCostQuickly(allBooks);

        cout << "Enter Author's name: ";
        string input;
        getline(cin, input);
        ShowSearchResult(byAuthor, input);

        cout << "Enter keyword: ";
        getline(cin, input);
        ShowSearchResult(byKeyword, input);

        cout << "Enter title of book: ";
        getline(cin, input);
        ShowBookCost(titleToCost, input);

        // de-allocate all Book objects
        CleanUp(allBooks);
}
vector<Book*> ReadFromDB()
{
	// simulate reading a lot of books from DB. We need to use Free Store instead of stack
	vector<Book*> books;
	books.push_back(new Book{ "Who Controls the Internet?", "Jack Goldsmith,Tim Wu",
"internet,controll", 1600 });
	books.push_back(new Book{ "Free Software, Free Society", "Richard M. Stallman",
"software,society,impact on society,free software", 800 });
	books.push_back(new Book{ "Innovation Happens Elsewhere", "Richard P. Gabriel",
"innovation,software,thinking in software", 700 });
	books.push_back(new Book{ "Patterns of Software", "Richard P. Gabriel",
"software,patterns,software engineering", 1100 });
	books.push_back(new Book{ "The Art of Computer Programming", "Donald Knuth",
"computer,programming,programming concepts", 650 });
	books.push_back(new Book{ "The Art of Unix Programming", "Eric Raymond",
"unix,programming,linux", 600 });
	books.push_back(new Book{ "Close to the Machine", "Ellen Ullman", "machine,man-
machine,modern world,impact of machines", 350 });

	return books;
}

// utility function to display a book's info
void PrintBookInfo(const Book* book)
{
	cout << "    Title: " << book->title << endl;
	cout << "Author(s): " << book->authors << endl << endl;
}
```

Q15. Complete the function below

```
void CleanUp(vector<Book*> allBooks)
{
	// deallocate all memory allocated by ReadFromDB
```

(2)

```
}
```

Q16. Complete the function below

```
void ListAllBooks(const vector<Book*> books)
{
	// display all books - use the PrintBookInfo() utility function
```

(2)

```
        cin.get();          // wait for user to hit a key
        system("cls"); // clear screen
}
```

Q17. Complete the function below
// the authors in Book object could be comma separated list. So could be keywords.
// so we write a common function to extract list of sub-strings comma separated in given string

```
vector<string> SplitCommaSeparatedList(………………………………………… original)
{                                                                                     (4)
        // you can assume there are no spaces around comma in original
        // i.e. original is like "a,b c,d" and not like "a, b c, d"
        vector<string> list;

        if (……………………………………………………………………….) { // if there are no commas
                list.push_back(original); // add the only substr to list
        }
        else { // if there are one or more commas
                size_t startPos = 0, pos = 0;
                for(;;) {
                        auto pos = ……………………………………………….. // find a comma

                        if (pos == ………………………………….) { // check for end
                                // if no more commas, we need to copy the last author

                                if (startPos != ……………………………………….) {

                                  list.push_back(……………………………………………………………….);
                                }
                                break; // break out of loop because we've reached the end
                        }
                        // add the extracted author to list
                        list.push_back(………………………………………………………….);
                        startPos = pos + 1;

                }
        }

        return list;
}
```

Q18. Complete the two functions below

(2+2)

```
………………………………………… IndexBooksToSearchByAuthor(…………………………………………
allBooks)
{
        ………………………………………………………….. authorToBooksMap;
        // iterate through all books and extract authors

        for (……………………………………) { // for each book

                auto authors = SplitCommaSeparatedList(……………………………………);

                for (……………………………) // for each author
                        // add entry into the map

                        ……………………………………………………………………………………….
        }
        return authorToBooksMap;
}

……………………………………………….. IndexBooksToSearchByKeyword(……………………………………… allBooks)
{
        ………………………………………………………… keywordToBooksMap;
        // iterate through all books and extract keywords
        for (……………………………………) {
                auto keywords = SplitCommaSeparatedList(………………………………………);
                for (…………………………………)
                        // add entry into the map

                        ……………………………………………………….
        }

        return keywordToBooksMap;
}
```

Q19. Complete the function below

(2)

```
…………………………………………………. BuildTableToFindCostQuickly(…………………………………… allBooks)
{
        …………………………………………….. titleToCost;

        for (…………………………………..) {
                // add entry
```

```
                    ……………………………………………………….;
          }

          return titleToCost;
}
```

Q20. Complete the two functions below (2+2)

```
void ShowSearchResult(……………………………………………………. byAuthor, string& searchByAuthor)
{
          if (byAuthor……………………………(searchByAuthor) == 0) { // write function name
                    cout << "Not Found";
          }
          else {
                    auto p = byAuthor……………………………….(searchByAuthor); // write the function name

                    for (auto i = …………………..; ………………………………….; ++i) {

                              PrintBookInfo(……………………………….);
                    }
          }
          cin.get();
          system("cls");
}

void ShowBookCost(……………………………………………. titleToCost, string& input)
{
          if (titleToCost…………………………(input) == 0)
                    cout << "No such book found" << endl;
          else
                    cout << "Cost: " << ………………………………………. << " INR" << endl;
          cin.get();
          system("cls");
}
```

____