30 - 20191 2019

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;
}
```

ID - 20191209

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.

```
Void PrintSubString(string_view fullString, int start, int count)
{
    if (fullString.length() >= start + count){
        cout << fullString.substr(start, count);
    }
}
```

(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)
{
```

ID - 2019 12019

```
        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)

```
Struct CompareByCpi {
bool operator()(Const Student& s1, const Student& s2)
    {
        return s1.cpi < s2.cpi;
    }
};
```

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();
priority_queue<Student, vector<student>, CompareByCpi> sq;
    for(auto s: v) {
        sq.push(s); }
2   while(!sq.empty()) {
        PrintStudentInfo(sq.top());
    } sq.poll();
```

7

ID - 201912019

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();
    deque<Student> dq;
    for (auto s : v) {
        if (s.gender == female)
            dq.push_front(s);
        else
            dq.push_back(s);
    }
    for (auto itr = dq.begin(); itr != dq.end(); itr++)
        PrintStudentInfo(*itr);
}
```

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;

    map<Gender, vector<Student>> groups;        // create map named groups
    map[female] = females;                      // assign females vector to female group
    map[male] = males;                          // assign males vector to male group

    for (auto s : v) { //iterate over v using range based for loop
        // based on gender put the student into appropriate group
        map[s.gender].push_back(s);
    }
    cout << "*** Female Students ***" << endl;

    for (auto itr = females.begin(); itr != females.end(); ++itr)
        PrintStudentInfo(*itr);
```

1.5

ID-20191209

```
    cout << "*** Male Students ***" << endl;
    for (auto itr = males.begin(); itr != males.end(); ++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();
    unordered_multiset<Student> cpis;
    for (auto s : v) {
        cpis.insert(s);
    }

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

    int count = cpis.count(checkCpi); // write the function call
    cout << "Students with CPI " << checkCpi.cpi << " = " << count << endl;

    auto p = cpis.equal_range(checkCpi);
    for (auto itr = p.first; itr != p.second; ++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
};
```

ID - 20191209

```
        // 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)
{ [2]

```
        // deallocate all memory allocated by ReadFromDB
for(int i=0; i<allBooks.size(); i++){
        delete allBooks[i];
}       allBooks[i] = nullptr;
}
```

Q16. Complete the function below
void ListAllBooks(const vector<Book*> books)
{

```
        // display all books - use the PrintBookInfo() utility function
for (auto b : books)
        PrintBookInfo(b);
```

[2]

ID - 20191 2019

```
        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(const String& original)
{
        // 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 (original.find(",") == string::npos) { // 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 = original.find(",", startPos);// find a comma

                        if (pos == string::npos) { // check for end
                                // if no more commas, we need to copy the last author
                                if (startPos != original.length()) {

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

                }
        }

        return list;
```

(4)

ID - 20191 2019

(2+2)

Q18. Complete the two functions below

```
multimap<string, Book*> IndexBooksToSearchByAuthor(vector<Book*> allBooks)
{
    multimap<string, Book*> authorToBooksMap;
    // iterate through all books and extract authors
    for (auto bk : allBooks) { // for each book
        auto authors = SplitCommaSeparatedList(bk->authors);
        for (auto s : authors) // for each author
            // add entry into the map
            authorToBooksMap.emplace(s, bk);
    }
    return authorToBooksMap;
}

multimap<string, Book*> IndexBooksToSearchByKeyword(vector<Book*> allBooks)
{
    multimap<string, Book*> keywordToBooksMap;
    // iterate through all books and extract keywords
    for (auto bk : allBooks) {
        auto keywords = SplitCommaSeparatedList(bk->keywords);
        for (auto k : keywords)
            // add entry into the map
            keywordToBooksMap.emplace(k, bk);
    }
    return keywordToBooksMap;
}
```

Q19. Complete the function below

(2

```
unordered_map<string, unsigned int> BuildTableToFindCostQuickly(vector<Book*> allBooks)
{
    unordered_map<string, unsigned int> titleToCost;
    for (auto book : allBooks) {
        // add entry
```

ID - 20191209

titleToCost[book->title] = book->cost;
}

return titleToCost;
}

Q20. Complete the two functions below

void ShowSearchResult(multimap<string, Book*> byAuthor, string& searchByAuthor) (2+2)
{
if (byAuthor.count(searchByAuthor) == 0) { // write function name
cout << "Not Found";
}
else {
auto p = byAuthor.equal_range(searchByAuthor); // write the function name
for (auto i = p.first; i != p.second; ++i) {
PrintBookInfo(*(i->second));
}
}
cin.get();
system("cls");
}

void ShowBookCost(unordered_map<string, unsigned int> titleToCost, string& input)
{
if (titleToCost.count(input) == 0)
cout << "No such book found" << endl;
else
cout << "Cost: " << titleToCost[input] << " INR" << endl;
cin.get();
system("cls");
}