SC1008 – C and C++ Programming

Freshmen Exemption Test Information

Topics

1. **Basic Constructs in C Language**
   C program structure. Syntax and semantics. Intrinsic data types, declarations, operators, assignments, control flow, and simple input/output. Pre-processing. Functions. Return values, arguments and parameter passing. Scopes of variables. Concept of side effects.

2. **Built-in Data Structures**
   Pointers, pointer operations and pass by reference. One-dimensional and multi-dimensional arrays, and pointers and arrays. Character strings and arrays of strings. Structures, arrays of structures and type definitions.

3. **Recursion**
   Basic recursion. Problem solving with recursion. Recursive versus iterative functions.

4. **Link Lists**
   Dynamic memory allocation. Linked list structures and applications.

5. **Basic C++ Programming**
   C++ Program Structure; Operators and expressions; Types and conversions; Function; Scope; Simple input/output; References; Strings in C++.

6. **Class and Object in C++**
   Class definition; Constructors and destructors; Class methods; Static class members; Access control or encapsulation; Structures and Unions.

7. **STL Basics in C++**
   Arrays; vectors.

8. **Inheritance and Polymorphism in C++**
   Inheritance; Method overloading and overriding; Multilevel inheritance; Multiple inheritance; Virtual functions; Polymorphism.

Teaching Schedule

Week	Topics	Readings	Delivery Mode	Activities
1	Basic C Programming (C program structure; Syntax and semantics; Intrinsic data types, declarations, operators, assignments, and simple input/output.)	Lecture notes and reference books	In-person	Lecture/Tutorial and Lab
2	Control Constructs (Branching constructs; Looping constructs; Procedure/function: activation, parameter passing, scopes of	Lecture notes and reference books	In-person	Lecture/Tutorial and Lab

	variables, and concept of side effects.)			
3	Pointers (Pointer types; Pointer operations; Pass by reference.)	Lecture notes and reference books	In-person	Lecture/Tutorial and Lab
4	Arrays (One-dimensional arrays; Multi-dimensional arrays.)	Lecture notes and reference books	In-person	Lecture/Tutorial and Lab
5	Character Strings (String constants and variables; String input/output; Character functions; String functions; Arrays of strings.)	Lecture notes and reference books	In-person	Lecture/Tutorial and Lab
6	Structures (Type definitions; Structured types; Arrays of Structures; Nested structures; Enumerated types.)	Lecture notes and reference books	In-person	Lecture/Tutorial and Lab
7	Recursions (Basic recursion; Problem solving with recursion; Recursive versus iterative functions.)	Lecture notes and reference books	In-person	Lecture/Tutorial and Lab
8	Linked Lists (Static vs dynamic memory allocation; Overview of node-based data structures; Linked list structures and applications.)	Lecture notes and reference books	In-person	Lecture/Tutorial and Lab

References

The course will not use any specific textbook. The following books will be used as reference materials.

1. S.C. Hui, "Introduction to C Programming", Pearson Education South Asia Pte Ltd, 2002.
2. Tony Gaddis, Judy Walters, Godfrey Muganda, "Starting Out with C++: Early Objects (6th Edition)", Pearson, 2016. (Chapter 17)
3. Prata, Stephen, "C++ primer plus (5th Edition)". Sams Publishing, 2002. (Chapters 1-4, 8, 10-13, 16)
4. Tony Gaddis, "Starting out with C++ from control structures through objects (8th edition)", 2014. (Chapters 15-17)

Exemption Test - Sample C Questions

1. **(digitPos)** Write the function **digitPos1()** that returns the position of the first appearance of a specified *digit* in a positive number *num*. The position of the digit is counted from the right and starts from 1. If the required digit is not in the number, the function should return 0. For example, digitPos1(12315, 1) returns 2 and digitPos1(12, 3) returns 0. The function prototype is given below:

```
int digitPos1(int num, int digit);
```

Write another function **digitPos2()** that passes the result through the pointer parameter, *result*. For example, if num = 12315 and digit = 1, then *result = 2 and if num=12 and digit = 3, then *result = 0. The function prototype is given below:

```
void digitPos2(int num, int digit, int *result);
```

The following sample program template is given for testing the functions:

```
#include <stdio.h>
int digitPos1(int num, int digit);
void digitPos2(int num, int digit, int *result);
int main()
{
   int number, digit, result=0;
   printf("Enter the number: \n");
   scanf("%d", &number);
   printf("Enter the digit: \n");
   scanf("%d", &digit);
   printf("digitPos1(): %d\n", digitPos1(number, digit));
   digitPos2(number, digit, &result);
   printf("digitPos2(): %d\n", result);
   return 0;
}
int digitPos1(int num, int digit)
{
   /* Write your code here */
}
void digitPos2(int num, int digit, int *result)
{
   /* Write your code here */
}
```

Some sample input and output sessions are given below:

(1) Test Case 1:
Enter the number:
234567
Enter the digit:
6
digitPos1(): 2
digitPos2(): 2

(2) Test Case 2:
Enter the number:
234567
Enter the digit:
8
digitPos1(): 0
digitPos2(): 0

2. **(reduceMatrix2D)** A square matrix (2-dimensional array of equal dimensions) can be reduced to upper-triangular form by setting each diagonal element to the sum of the original elements in that column and setting to 0s all the elements below the diagonal. For example, the 4-by-4 matrix:

```
4  3  8  6
9  0  6  5
5  1  2  4
9  8  3  7
```

would be reduced to

```
27  3  8  6
 0  9  6  5
 0  0  5  4
 0  0  0  7
```

Write a function reduceMatrix2D() to reduce a matrix with dimensions of *rowSize* and *colSize*. The prototype of the function is:

```
void reduceMatrix2D(int ar[][SIZE], int rowSize, int colSize);
```

SIZE is a constant defined at the beginning of the program. For example, #define SIZE 10. The parameters *rowSize* and *colSize* are used to specify the dimensions of the 2-dimensional array (e.g., 4x4) that the function should process.

A sample program template is given below to test the function:

```
#include <stdio.h>
#define SIZE 10
void reduceMatrix2D(int ar[][SIZE], int rowSize, int colSize);
void display(int ar[][SIZE], int rowSize, int colSize);
int main()
{
  int ar[SIZE][SIZE], rowSize, colSize;
  int i,j;
  printf("Enter row size of the 2D array: \n");
  scanf("%d", &rowSize);
  printf("Enter column size of the 2D array: \n");
  scanf("%d", &colSize);
  printf("Enter the matrix (%dx%d): \n", rowSize, colSize);
  for (i=0; i<rowSize; i++)
    for (j=0; j<colSize; j++)
      scanf("%d", &ar[i][j]);
  reduceMatrix2D(ar, rowSize, colSize);
  printf("reduceMatrix2D(): \n");
  display(ar, rowSize, colSize);
  return 0;
}
void display(int ar[][SIZE], int rowSize, int colSize)
{
  int l,m;
  for (l = 0; l < rowSize; l++) {
    for (m = 0; m < colSize; m++)
      printf("%d ", ar[l][m]);
    printf("\n");
  }
}
void reduceMatrix2D(int ar[][SIZE], int rowSize, int colSize)
{
  /* Write your code here */
}
```

Some sample input and output sessions are given below:

(1) Test Case 1:
Enter row size of the 2D array:
4
Enter column size of the 2D array:
4
Enter the matrix (4x4):
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
reduceMatrix2D():
28 2 3 4
0 30 7 8
0 0 26 12
0 0 0 16

(2) Test Case 2:
Enter row size of the 2D array:
3
Enter column size of the 2D array:
3
Enter the matrix (3x3):
1 0 0
2 2 0
3 3 3
reduceMatrix2D():
6 0 0
0 5 0
0 0 3

(3) Test Case 3:
Enter row size of the 2D array:
4
Enter column size of the 2D array:
4
Enter the matrix (4x4):
1 2 3 4
7 8 9 10
5 6 7 8
11 12 13 14
reduceMatrix2D():
24 2 3 4
0 26 9 10
0 0 20 8
0 0 0 14

(4) Test Case 4:
Enter row size of the 2D array:
4
Enter column size of the 2D array:
4
Enter the matrix (4x4):
-5 -6 -7 -8
3 4 5 6
-1 -2 -3 -4
6 7 8 9
reduceMatrix2D():
3 -6 -7 -8
0 9 5 6
0 0 5 -4
0 0 0 9

3. **(stringcmp)** Write a C function that compares the string pointed to by *s1* to the string pointed to by *s2*. If the string pointed to by *s1* is greater than, equal to, or less than the string pointed to by *s2*, then it returns 1, 0 or –1 respectively. Write the code for the function without using any of the standard C string library functions. The function prototype is given as follows:

```
int stringcmp(char *s1, char *s2);
```

A sample template for the program is given below:

```
#include <stdio.h>
#include <string.h>
#define INIT_VALUE 999
int stringcmp(char *s1, char *s2);
int main()
{
   char source[80], target[80], *p;
   int result = INIT_VALUE;

   printf("Enter a source string: \n");
   fgets(source, 80, stdin);
   if (p=strchr(source,'\n')) *p = '\0';
   printf("Enter a target string: \n");
   fgets(target, 80, stdin);
   if (p=strchr(target,'\n')) *p = '\0';
   result = stringcmp(source, target);
   if (result == 1)
      printf("stringcmp(): greater than");
   else if (result == 0)
      printf("stringcmp(): equal");
   else if (result == -1)
      printf("stringcmp(): less than");
   else
      printf("stringcmp(): error");
   return 0;
}
int stringcmp(char *s1, char *s2)
{
   /* Write your code here */
}
```

Some test input and output sessions are given below:

(1) Test Case 1:
Enter a source string:
abc
Enter a target string:
abc
stringcmp(): equal

(2) Test Case 2:
Enter a source string:
abcdefg
Enter a target string:
abcde123
stringcmp(): greater than

(3) Test Case 3:
Enter a source string:
abc123
Enter a target string:
abcdef
stringcmp(): less than

(4) Test Case 4:
Enter a source string:
abcdef

Enter a target string:
abcdefg
stringcmp(): less than

4. (**rCountArray**) Write a **recursive** C function rCountArray() that returns the number of times the integer *a* appears in the array which has *n* integers in it. Assume that *n* is greater than or equal to 1. The function prototype is:

```
int rCountArray(int array[], int n, int a);
```

A sample C program is given below to test the function:

```
#include <stdio.h>
#define SIZE 20
int rCountArray(int array[], int n, int a);
int main()
{
   int array[SIZE];
   int index, count, target, size;

   printf("Enter array size: \n");
   scanf("%d", &size);
   printf("Enter %d numbers: \n", size);
   for (index = 0; index < size; index++)
      scanf("%d", &array[index]);
   printf("Enter the target number: \n");
   scanf("%d", &target);
   count = rCountArray(array, size, target);
   printf("rCountArray(): %d\n", count);
   return 0;
}
int rCountArray(int array[], int n, int a)
{
   /* Write your program code here */
}
```

Some sample input and output sessions are given below:

(1) Test Case 1:
Enter array size:
10
Enter 10 numbers:
1 2 3 4 5 5 6 7 8 9
Enter the target number:
5
rCountArray(): 2

(2) Test Case 2:
Enter array size:
5
Enter 5 numbers:
1 2 3 4 5
Enter the target number:
8
rCountArray(): 0

(3) Test Case 3:
Enter array size:
1
Enter 1 numbers:
5
Enter the target number:
5
rCountArray(): 1

(4) Test Case 4:

```
Enter array size:
7
Enter 5 numbers:
1 2 3 3 4 3 3
Enter the target number:
3
rCountArray(): 4
```

Exemption Test - Sample C++ Questions

1. Write **two template functions** `calArea()` in C++ to calculate the area of a rectangle and a trapezoid. The input parameter type of the function can be any types of numbers, e.g., int, double, float, long. All the parameters of the function are of the same type. The detailed math equations are as follows.

a) The area of a rectangle:

$$A = l \times w$$

where l is the length of the rectangle and w is the width of the rectangle.

b) The area of a trapezoid:

$$A = \frac{1}{2}(b_1 + b_2)h$$

where b_1 and b_2 are the top and bottom base lengths, and h is the height.

2. Write C++ code to convert an array of strings (e.g., student names) to a linked list. The structure of each node in the linked list is defined as follows:

```
struct StringNode {
    string name;
    StringNode* next;
};
```

You are asked to insert elements of the input array one by one at the beginning of the linked list, instead of at the end of the linked list. The function prototype is as follows:

```
void arrayToLinkedList(const string* arr, int size, StringNode*& head);
```

It is possible that the input string array is empty (i.e., arr is empty and size is 0).
Below is the code for the main function with two test cases and other relevant functions.

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

// Define the structure of a linked list node
struct StringNode {
    string name;
    StringNode* next;
};

// Function to print the linked list
void printList(StringNode* head) {
        StringNode* temp = head;
    cout << "Linked list: ";
    while (temp) {
        cout << temp->name << " -> ";
        temp = temp->next;
```

```
    }
    cout << "NULL" << endl;
}

// Function to free allocated memory
void deleteList(StringNode*& head) {
    while (head) {
        StringNode* temp = head;
        head = head->next;
        delete temp;
    }
    head = nullptr;
}

// To-do: Create a linked list from an array of strings
void arrayToLinkedList(const string* arr, int size, StringNode*& head) {
    // Please write down your code here
    //
    //


}

int main() {
    // Case 1
    string students[] = {"Alice", "Bob", "Charlie", "David"};
    int size = sizeof(students) / sizeof(students[0]);
    StringNode* head1 = nullptr;
    arrayToLinkedList(students, size, head1);
    printList(head1);

    // Case 2
    string companyNames[] = {"Microsoft", "Google", "Tencent", "Alibaba", "HP"};
    size = sizeof(companyNames) / sizeof(companyNames[0]);
    StringNode* head2 = nullptr;
    arrayToLinkedList(companyNames, size, head2);
    printList(head2);

    deleteList(head1);
    deleteList(head2);
    return 0;
}
```

Sample output should be:

```
Linked list: David -> Charlie -> Bob -> Alice -> NULL
Linked list: HP -> Alibaba -> Tencent -> Google -> Microsoft -> NULL
```

3. You are asked to implement a class called Pen to manage the records of pens in an online shopping platform. It allows the setting of the color and price of a pen, as well as display them. Below are the detailed members of this class.

- Private members
    - std::string color: the color of the pen

- double price: the price of the pen

- Public members
  - Constructor that initializes the color to black and price to 10
  - setColor(), the method to set the color of the pen
  - setPrice(), the method to set the price of the pen
  - display(), the method to display the pen details

The last three public member functions above should be able to enable method chaining.
The main function and the starting code of the Pen class are given below.

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

class Pen {
private:
   string color;
   double price;

public:
   // TO-DO: Implement your constructor here



   Pen& setColor(string newColor) {
      // TO-DO: Write your code here
      //

   }

   Pen& setPrice(double newPrice) {
      // TO-DO: Write your code here
      //

      }

   void display() const {
      // TO-DO: Write your code here
      //

   }
};



int main() {
   // Case 1: Default Pen
   Pen myDefaultPen;
   cout<< "The pen information: " << endl;
   myDefaultPen.display();

   // Case 2: A pen with configuration
   Pen myPen("Blue", 1.5);
   cout<<endl;
```

```
    cout<< "The pen information: " << endl;
    myPen.display();

    // Case 3: Modify the pen via method chaining
    cout<<endl;
    cout<< "The pen information: " << endl;
    myPen.setColor("Red")
        .setPrice(2.0)
        .display();

    return 0;
}
```

Sample output should be:

```
The pen information:
Pen Color: black
Price: $10

The pen information:
Pen Color: Blue
Price: $1.5

The pen information:
Pen Color: Red
Price: $2
```

4. Create a base class Person with attributes for name and age. Then, create a derived class Student that adds a studentID attribute.

The details for members of the two classes are as follows:

Base Class Person

- Attributes:
  - string name
  - int age
- Constructor: Person(string n, int a)
- Function: void displayInfo() const

Derived Class Student (inherits from Person)

- Attribute: studentID
- Constructor: Student(string n, int a, int id)
- Function: void displayInfo() const (**override** the base class function)

Below is the starting code with missing implementation for you to fill in:

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

// Base Class: Person
```

```
class Person {
// TO-DO: Implement the Person Class here

};

// Derived Class: Student
class Student : public Person {
// TO-DO: Implement the Student Class here

};

// main()
int main() {
    Person p1("Bob", 18);
    p1.displayInfo();
    cout << endl;

    Student s1("Alice", 20, 12345);
    s1.displayInfo();
    cout << endl;

    Person* p = &s1;
    p->displayInfo();

    return 0;
}
```

Sample output:

```
Name: Bob
Age: 18

Name: Alice
Age: 20
Student ID: 12345

Name: Alice
Age: 20
Student ID: 12345
```

5. A small café wants to track its daily sales. Each day's sales figure is an integer. You are asked to use an STL vector container, i.e., std::vector<int> dailySales to track the daily sales. You are requested to implement the following functions:

- AddSales(): Insert an array of sale values into the vector.
- PrintSales(): Traverse and print all sales values using an iterator. Each value is separated by an empty space
- CalAverage(): Compute the average sales.
- SortSales(): Sort the sales in ascending order. Hint: you can consider using std::sort().

Note: It is possible that the vector container is empty.

Here is the sample code for you to get started:

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <numeric>
using namespace std;

// TO-DO: Implement the AddSales() function here
void AddSales(vector<int>& dailySales, const int sales[], int size) {

}


// TO-DO: Implement the PrintSales() function here, with each value separated by an empty space
void PrintSales(const vector<int>& dailySales) {


}

// TO-DO: Implement the CalAverage() function here
double CalAverage(const vector<int>& dailySales) {


}

// TO-DO: Implement the SortSales() function here
void SortSales(vector<int>& dailySales) {



}

int main() {
    vector<int> dailySales;

    int arr[] = {120, 80, 150, 90, 200};
    int size = sizeof(arr) / sizeof(arr[0]);

    AddSales(dailySales, arr, size);

    cout << "Before sorting:" << endl;
    PrintSales(dailySales);

    SortSales(dailySales);
    cout << "After sorting:" << endl;
    PrintSales(dailySales);

    cout << "Average sales: " << CalAverage(dailySales) << endl;

    return 0;
}
```

Expected output:

```
Before sorting:
120 80 150 90 200
After sorting:
80 90 120 150 200
Average sales: 128
```