Programming in Modern C++: Assignment Week 0

Total Marks : 30

Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur 721302
partha.p.das@gmail.com, ppd@cse.iitkgp.ac.in

June 14, 2024

Question 1

Consider the below code snippet. *[MCQ, Marks 2]*

```
#include <stdio.h>

int main() {

    typedef double num[3];

    num array[5] = {1,2,3,4,5,6}; // LINE-1

    printf("%u", sizeof(array));

    printf(" %.2f", array[1][1]);

    return 0;
}
```

What will be the output/error (`sizeof(double) = 8 bytes`)?

a) `40 2.00`

b) `120 5.00`

c) `120 2.00`

d) `Compilation error at LINE-1`

Answer: b)
Explanation:
`typedef` is a keyword used in C language to assign alternative names to existing data types. Here, `num array[5]` is equivalent to `double array[5][3]`. So, `sizeof(array)` = 5 * 3 * 8 = 120 bytes.

The structure of array looks like `array[5][3] = {{1.00, 2.00, 3.00}, {4.00, 5.00, 6.00}, {0.00, 0.00, 0.00}, {0.00, 0.00, 0.00}, {0.00, 0.00, 0.00}}`.

Hence, `array[1][1]` gives output 5.00.

Note: Default initialization of the array element is 0.00 though it varies from compiler to compiler.

Question 2

Consider the following code segment. *[MCQ, Marks 2]*

```
#include <stdio.h>

enum Covid_prevention {
    Sanitizer = 1,
    Wear_mask = 2,
    Soc_distance = 4
};

int main() {
    int myCovidPrevention = Wear_mask | Soc_distance;

    printf("%d", myCovidPrevention);

    return 0;
}
```

What will be the output?

a) `2`

b) `4`

c) `6`

d) `8`

Answer: c)
Explanation:
The `enum` value `Wear_mask` and `Soc_distance` are assigned with a **bitwise OR** (The pipe symbol, `|`) operator to the integer variable `myCovidPrevention`. So value of `Wear_mask | Soc_distance = 2 | 4 = 000010 | 000100 = 000110 = 6.`

Note that, int is of 4 bytes though we are considering the right most six bits for calculation and ignore all zeros in the left.

Question 3

Consider the below program. *[MCQ, Marks 2]*

```
#include <stdio.h>

int main() {
    int x = 1;
    switch(x)
    {
        case x:
            printf("case 1 ");
            break;
        case x + 1;
            printf("case 2 ");
            break;
        default:
            printf("default block");
            break;
    }
    return 0;
}
```

What will be the output/error?

a) `case 1`

b) `case 2`

c) `default block`

d) `Compilation error`: `'x'` expected to be an integer or a character constant

Answer: d)
Explanation:
The `case` statement accepts only an `int` or `char` constant expression. As the case statement is having variable i.e. not constant expression, the program will give compilation error.

Question 4

Consider the following linked list: *[MCQ, Marks 2]*

```
I -> I -> T -> K -> G -> P
```

What is the output of the following function when it is called with the head of the list?

```
void fun(struct node* start) {
    if (start == NULL)
        return;
    printf("%c ", start->data); // Considering data is of 'char' type
    if (start->next != NULL)
        fun(start->next->next);
    printf("%c ", start->data);
}
```

a) `I T G I G`

b) `I T G G`

c) `I T G G T I`

d) `I T G I T G`

Answer: c)
Explanation:
`fun()` prints alternate nodes of the given Linked List, first from head to end, and then from end to head. If the Linked List has an even number of nodes, then skips the last node.

Question 5

A single array `A[1..MAXSIZE]` is used to implement two stacks. The two stacks grow from opposite ends of the array. Variables `t1` and `t2 (tl < t2)` point to the location of the topmost element in each of the stacks. If the space is to be used efficiently, the condition for stack full is: *[MCQ, Marks 2]*

a) `(t1 = MAXSIZE/2) and (t2 = MAXSIZE/2+1)`

b) `t1 + t2 = MAXSIZE`

c) `(t1 = MAXSIZE/2) or (t2 = MAXSIZE)`

d) `t1 = t2 - 1`

Answer: d)
Explanation:
If we are to use space efficiently, then the size of the any stack can be more than `MAXSIZE/2`. Both stacks will grow from both ends and if any of the stack top reaches near to the other top then the stacks are full. So the condition will be `t1 = t2 -1` (given that `t1` $\leq$ `t2`)

Question 6

Consider the following code segment. *[MCQ, Marks 2]*

```
void fun(Queue *que) {

    Stack Stk;

    while (!isEmpty(que)) {
        push(&Stk, deQueue(que));
    }

    while (!isEmpty(&Stk)) {
        enQueue(que, pop(&Stk));
    }
}
```

where `push` and `pop` are two standard functionalities of stack data structure. Similarly, `enQueue` and `deQueue` are two standard functionalities of queue data structure to insert and delete the items respectively. And `isEmpty` checks if the stack or the queue is empty or not.

What does the above function do?

a) Remove the last element from `que`

b) Reverse the elements in the `que`

c) Keeps the `que` unchanged

d) Makes `que` empty

Answer: b)
Explanation:
The function takes a queue `que` as an argument. It dequeues all items of `que` and pushes them to a stack `Stk`. Then pops all items of `Stk` and enqueues the items back to `que`. Since stack is LIFO order, all items of queue are reversed.

Question 7

Consider the below code segment. *[MCQ, Marks 2]*

```
#include <stdio.h>

int main() {
    int x = 1;
    int y;

    y = (x=x+5, x*5);

    printf("%d",y);

    return 0;
}
```

What will be the output?

a) `25`

b) `30`

c) `6`

d) `5`

Answer: b)
Explanation:
Comma operator evaluates multiple expressions from left to right. The value of rightmost (`x*5`) expression is assigned to `y`. Here first `x=x+5` will be evaluated and make the value of x as 6. Then second expression `x*5` will be evaluated and 6*5=30 will be assigned to `y`.

Question 8

Consider a three-dimensional array `arr[5][10][20]`. An element from this array can be represented as `arr[i][j][k]` where $0 \leq i \leq 4$, $0 \leq j \leq 9$ and $0 \leq k \leq 19$. How can you write `arr[2][6][10]` in an equivalent pointer expression? *[MSQ, Marks 2]*

a) `((**(*a+2)+6)+10)`

b) `(**(*(a+2)+6)+10)`

c) `(*(**(a+2)+6)+10)`

d) `*(*(*(a+2)+6)+10)`

Answer: d)
Explanation:
C represents an array as row-major. As a multidimensional array is stored in one dimensional fashion in memory, the innermost index is the slowest to change. Hence, the equivalent pointer which points to the element `arr[i][j][k]` is `*(*(*(a+i)+j)+k)`. Hence, the correct option is d).

Question 9

Consider the code segment below. *[MCQ, Marks 2]*

```
#include <stdio.h>

int main() {
    int *p, n = 5;
    p = &n;
    *p += 1;

    printf("%d,%d", *p, n);

    return 0;
}
```

What will be the output?

a) `5,5`

b) `5,6`

c) `6,5`

d) `6,6`

Answer: d)
Explanation:
The address of variable `n` is assigned to the pointer variable `p`. So, whatever changes are done in pointer variable will be reflected to `n`. The value of `*p` is incremented by 1. So, the value of `n` will also be incremented by 1. Hence, output will be `6,6`.

Question 10

Consider the code segment below. *[MCQ, Marks 2]*

```
#include <stdio.h>

struct result {
    char subject[20];
    int mark;
};

int main() {
    struct result r[] = {
        {"Maths",95},
        {"Science",93},
        {"English",80}
    };
    printf("%s ", r[1].subject);
    printf("%d", (*(r+2)).mark);

    return 0;
}
```

What will be the output?

a) `Science 80`

b) `Science 93`

c) `English 80`

d) `English 93`

Answer: a)
Explanation:
`r` is an array variable of structure `result` type. The first print statement will print `subject` value of the second array item. Similarly, the second print statement will print `mark` value of the third array element. Hence, the correct option is a).

Question 11

Consider the code segment below. *[MCQ, Marks 2]*

```
#include <stdio.h>

void teller1(char* msg) {
    printf("teller1: %s\n", msg);
}
void teller2(char* msg) {
    printf("teller2: %s\n", msg);
}
void teller3(char* msg) {
    printf("teller3: %s\n", msg);
}

__________________________          // LINE-1
void caller(char *msg, F_PTR fp) {
    fp(msg);
}

int main() {
    caller("Hello", &teller1);
    caller("Hi", &teller2);
    caller("Good Morning", &teller3);

    return 0;
}
```

Identify the correct option to fill in the blank at `LINE-1`, such that the output is:

```
teller1: Hello
teller2: Hi
teller3: Good Morning
```

a) `void (*F_PTR)(char*);`

b) `typedef void (*F_PTR)(char*);`

c) `void *F_PTR(char*);`

d) `typedef void (*f_ptr)(char*) F_PTR;`

Answer: b)
Explanation:
Since `F_PTR` is the name of the function pointer, we need to define it with `typedef`. Thus, the correct option is b).

Question 12

Consider the code segment below. *[MSQ, Marks 2]*

```
#include <stdio.h>

int main() {
    int array[] = {10, 20, 30, 40, 50};
    int *ip, i;

    for(ip = array + 4, i = 0; i < 5; i++)
        printf("%d ", ________);    // LINE-1

    return 0;
}
```

Identify the correct option/s to fill in the blank at `LINE-1`, such that the output is:

```
50 40 30 20 10
```

a) `-i[ip]`

b) `ip[-i]`

c) `-ip[i]`

d) `(-i)[ip]`

Answer: b), d)
Explanation:
`-i[ip]` is equivalent to `-*(i + ip)`, which prints `-50` followed by 4 garbage values. So it is wrong option.
`ip[-i]` is equivalent to `*(ip - i)`, which prints `50 40 30 20 10`.
`-ip[i]` is equivalent to `-*(i + ip)`, which prints `-50` followed by 4 garbage values. So it is wrong option.
`(-i)[ip]` is equivalent to `*(-i + ip)`, which prints `50 40 30 20 10`.

Question 13

Consider the code segment below. *[MCQ, Marks 2]*
Assume that the `sizeof(int) = 4`

```
#include <stdio.h>

union uData {
    int a;
    int b;
};

struct sData {
    union uData c;
    int d;
};

int main() {
    struct sData da = {10, 20};

    printf("%ld ", sizeof(da));
    printf("%d %d %d", da.c.a, da.c.b, da.d);

    return 0;
}
```

What will be the output?

a) `8 10 10 20`

b) `16 10 20 <garbage-value>`

c) `16 10 <garbage-value> 20`

d) `8 10 <garbage-value> 20`

Answer: a)
Explanation:
`sizeof(union uData) = 4`, since the size of a `union` is same as the size of the largest element of the `union`
Thus, the `sizeof(da) = sizeof(struct sData) = sizeof(union uData) + sizeof(d) = sizeof(union uData) + sizeof(int) = 8`.
10 is intialized to the union data member. Hence, both union data member will hold 10 when accessed. Hence, `da.c.a = 10`, `da.c.b = 10` and `da.d = 20`.

Question 14

Consider the code segment below. *[MCQ, Marks 2]*

```
#include <stdio.h>

int main() {
    int x = 8, y, z;

    y = --x;
    z = x--;

    printf("%d %d %d", x, y, z);

    return 0;
}
```

What will be the output?

a) `8 7 7`

b) `8 7 6`

c) `6 7 7`

d) `6 7 6`

Answer: c)
Explanation:
The first expression is `y = --x`; so, `x` becomes 7 because of pre-decrement operator and value of `y = 7`. in the next step, the post decrement operator is applied on `x`. So, current value of `x` is assigned to z (`z = 7`) then `x` decremented by 1 (`x = 6`). Hence, final values will be `x=6, y=7 and z=7`.

Question 15

Consider the code segment below. *[MCQ, Marks 2]*

```
#include <stdio.h>

int main() {
    int p = 5, q = 6;

    printf("%d ", ++(p+q+5));

    return 0;
}
```

What will be the output/error?

a) `35`

b) `36`

c) `41`

d) `Compilation error:  lvalue required as increment operand`

Answer: d)
Explanation:
The operand of unary operator (increment / decrement) must be a variable, not a constant or expression. In our case, unary increment operator is applied on expression `p+q+5` which throws a compilation error.

Programming in Modern C++: Assignment Week 1

Total Marks : 25

Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur, Kharagpur – 721302
partha.p.das@gmail.com

June 25, 2024

Question 1

Consider the following program. *[MCQ, Marks 1]*

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

int main() {
    string message = "Good Morning World";
    _________________;    // LINE-1
    cout << message;
    return 0;
}
```

Fill in the blank at `LINE-1` such that the output is `Good Morning`.

a) `message.resize(12)`

b) `message.clear()`

c) `message.replace(0, 12, "Good Morning")`

d) `strcpy(message, "Good Morning")`

Answer: a)
Explanation:
`message.resize(12)` function resizes the string to the length of 12, leaving `Good Morning`.
`message.clear()` function erases the contents of the string, leaving it empty.
`message.replace(0, 12, "Good Morning")` function replaces the first 12 characters of the string with `"Good Morning"`, but since the first 12 characters are already "Good Morning", it does not change the string.
`strcpy(message, "Good Morning")` function gives a compilation error as `strcpy` can only be applied to C-strings, not C++ strings.
Hence, the correct option is a).

Question 2

Consider the following code segment. *[MSQ, Marks 1]*

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

bool compare(int i, int j) {
    return (i < j);
}

int main() {
    int arr[] = {9, 4, 2, 8, 6, 3, 1};
    sort(_______________________);    // LINE-1

    for(int i = 0; i < 7; i++)
        cout << arr[i] << " ";
    return 0;
}
```

Identify the appropriate option(s) to fill in the blank at `LINE-1`, such that the output is:
`2 4 9 8 6 3 1`

a) `&arr[0], &arr[0] + 3, compare`

b) `arr, arr + 3, compare`

c) `&arr[0], &arr[0] + 2, compare`

d) `arr, arr + 2, compare`

Answer: a), b)
Explanation:
As the output suggests, the array `arr` needs to be sorted up to the 3rd element. Hence, the `sort` function will take the first address of the array as the first argument and the address of the fourth element (i.e., one past the third element) as the second argument. Hence, the correct options are a) and b).

Question 3

Consider the following code segment. *[MCQ, Marks 2]*

```
#include <iostream>
#include <algorithm>
using namespace std;
int main () {
    int values[] = {60, 25, 35, 15, 45};
    sort (&values[1], &values[4]);
    for (int i = 0; i < 5; i++)
        cout << values[i] << " ";
    return 0;
}
```

What will be the output?

a) `60 15 25 35 45`

b) `60 25 15 35 45`

c) `60 25 35 45 15`

d) `60 25 35 15 45`

Answer: a)
Explanation:
Since the call is `sort(&values[1], &values[4])`, it considers 3 elements of the array `values[]` from the second element for sorting. Thus, it prints 60 15 25 35 45.

Question 4

Consider the following code segment. *[MCQ, Marks 2]*

```
#include <iostream>
#include <algorithm>
using namespace std;
int main() {
    int numbers[6];
    for(int i = 0; i < 6; i++)
        *(numbers + i) = (i + 1) * 10;
    rotate(numbers, numbers + 5, numbers + 6);
    rotate(numbers, numbers + 2, numbers + 5);
    for (int i = 0; i < 6; ++i)
        cout << numbers[i] << " ";
    return 0;
}
```

What will be the output?

a) `10 20 30 40 50 60`

b) `10 20 60 30 40 50`

c) `60 30 40 50 10 20`

d) `20 30 40 60 10 50`

Answer: d)
Explanation:
The `rotate(first, middle, last)` function rotates the order of the elements in the range `[first, last)`, such that the element pointed to by `middle` becomes the new first element. After the first rotation with `rotate(numbers, numbers + 5, numbers + 6)`, the array becomes {60, 10, 20, 30, 40, 50}. After the second rotation with `rotate(numbers, numbers + 2, numbers + 5)`, the array becomes {20, 30, 40, 60, 10, 50}. Hence, the output is `20 30 40 60 10 50`.

Question 5

Consider the following code segment. *[MCQ, Marks 2]*

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
    const int size = 4, c = 100;
    vector<int> vi(size, 5);
    for (int i = 1; i <= 3; i++)
        vi.push_back(c + i);
    vi.resize(12, 99);
    vi.resize(10);
    for (int i = 0; i < vi.size(); i++)
        cout << vi[i] << " ";
    return 0;
}
```

What will be the output?

a) `5 5 5 5 101 102 103 99 99 99`

b) `5 5 5 5 100 101 102 103 99 99`

c) `5 5 5 5 101 102 103 99 99`

d) `5 5 5 5 100 101 102 103 99 99 99 99`

Answer: a)
Explanation:
Vectors are similar to dynamic arrays having the ability to resize themselves automatically when an element is inserted or deleted, with their storage being handled automatically by the container. The statements and the states of the vector are as follows:

- `vector<int> vi(size, 5);` creates a vector with initial values [‘5’, ‘5’, ‘5’, ‘5’],
- `vi.push_back(c + i);` adds values [‘5’, ‘5’, ‘5’, ‘5’, ‘101’, ‘102’, ‘103’] (since 100 is the value of ‘c’ and i varies from 1 to 3),
- `vi.resize(12, 99);` changes the vector to [‘5’, ‘5’, ‘5’, ‘5’, ‘101’, ‘102’, ‘103’, ‘99’, ‘99’, ‘99’, ‘99’, ‘99’],
- `vi.resize(10);` reduces the size to [‘5’, ‘5’, ‘5’, ‘5’, ‘101’, ‘102’, ‘103’, ‘99’, ‘99’, ‘99’].

Question 6

Consider the following code segment. *[MCQ, Marks 2]*

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

int main() {
string str = "Programming";
str.erase(3, 5);
str.insert(3, "AB");
str.insert(5, "XYZ");
cout << str;
return 0;
}
```

What will be the output?

a) `ProABXYZing`

b) `ProABgXYZramming`

c) `ProABXYZmming`

d) `ProXYZABgramming`

Answer: a)
Explanation:
The initial string is "Programming". The `erase(3, 5)` function call removes 5 characters starting from index 3, resulting in "Proing". The `insert(3, ‘‘AB")` function call inserts "AB" at index 3, resulting in "ProABing". Finally, the `insert(5, "XYZ")` function call inserts "XYZ" at index 5, resulting in "ProABXYZing". Therefore, the correct output is "ProABXYZing".

Question 7

Consider the following code segment. *[MCQ, Marks 2]*

```
#include <iostream>
#include <algorithm>
using namespace std;
int main() {
    int array[] = { 10, 20, 30, 40, 50 };
    for (int i = 0; i < 1; i++) {
        int j = array[i];
        replace(array, array + 5, j, *(__________________));    //LINE-1
    }
    for (int i = 0; i < 5; ++i)
        cout << array[i] << " ";
    return 0;
}
```

Fill in the blank at `LINE-1` such that the output is
`50 20 30 40 50`

a) `array + 4 - i`

b) `array + 5 - i`

c) `array + i - 4`

d) `array + i - 5`

Answer: a)
Explanation:
The statement: `replace(array, array + 5, j, *(array + 4 - i));` replaces the first element (10) with the last element (50), resulting in the array `50 20 30 40 50`. Hence, `LINE-1` will be filled as `array + 4 - i` .

Question 8

Consider the following code segment. *[MCQ, Marks 2]*

```
#include <iostream>
#include <cstring>
#include <stack>

using namespace std;

int main(){
    char str[12] = "ABCDEFGHIJK";
    stack<char> stack1, stack2;
    int i;
    for(i = 0; i < strlen(str)/2; i++)
        stack1.push(str[i]);
    for(i=i-1; i < strlen(str); i++)
        stack2.push(str[i]);

    while (!stack1.empty()) {
        stack2.push(stack1.top()); stack1.pop();
    }
    while (!stack2.empty()) {
        cout << stack2.top(); stack2.pop();
    }
    return 0;
}
```

What will be the output?

a) `ABCDEKJIHGFE`

b) `ABCDEKJIHG`

c) `ABCDEJIHGF`

d) `ABCDEFGHIJK`

Answer: a)
Explanation:
The stack `stack1` stores {'A', 'B', 'C', 'D', 'E'} and the stack `stack2` stores {'E', 'F', 'G', 'H', 'I', 'J', 'K'}. Then the elements of `stack1` are also pushed into `stack2` in the order {'E', 'D', 'C', 'B', 'A'}. Thus, when we finally pop and print the elements from `stack2`, the output would be `ABCDEKJIHGFE`.

Question 9

Consider the following code segment. *[MCQ, Marks 2]*

```
int x = 10;
const int *a = &x;
int * const b = &x;
int const *c = &x;
int const * const d = &x;

*a = 20; //STMT-1
*b = 20; //STMT-2
*c = 20; //STMT-3
*d = 20; //STMT-4
```

Which statement/statements is/are correct?

a) `STMT-1`

b) `STMT-2`

c) `STMT-3`

d) `STMT-4`

Answer: b)
Explanation:
In statement `const int *a = &x;`, for `a` the pointee is constant, hence `*a` cannot be modified. So a) is incorrect.
In statement `int const *c = &x;`, again for c the pointee is constant, hence `*c` cannot be modified. So c) is incorrect.
In statement `int const * const d = &x;`, for d the pointer and pointee both are constant, hence `*d` cannot be modified. So d) is incorrect.
But in statement `int * const b = &x;`, for b the pointer is constant, but the pointee can be modified. Hence, `*b` can be modified. So b) is the correct option.

Programming Questions

Question 1

Consider the program below.

- Fill in the blank at `LINE-1` to declare a stack variable `st`.
- Fill in the blank at `LINE-2` to push values into the stack.
- Fill in the blank at `LINE-3` with the appropriate statement.

The program must satisfy the given test cases. *Marks: 3*

```
#include<iostream>
#include<cstring>
#include<stack>
using namespace std;
int main() {
    char input[20];
    char character;
    cin >> input;
    _________________; //LINE-1
    for(int i = 0; i < strlen(input); i++)
        ________________; //LINE-2
    for(int i = 0; i < strlen(input); i++) {
        character = _________;  //LINE-3
        cout << character;
        st.pop();
    }
    return 0;
}
```

Public 1

```
Input: coding
Output: gnidoc
```

Public 2

```
Input: reverse
Output: esrever
```

Private

```
Input: stack
Output: kcats
```

Answer:

```
LINE-1:  stack<char> st;
LINE-2:  st.push(input[i]);
LINE-3:  st.top();
```

Explanation:

To reverse the input string using a stack, we need to declare a stack of characters at `LINE-1` as `stack<char> st;`. At `LINE-2`, we push each character of the input string into the stack using `st.push(input[i]);`. At `LINE-3`, we retrieve and print the top element of the stack using `st.top();`. We then pop the top element of the stack until the stack is empty, effectively reversing the input string when printed.

Question 2

Consider the following program.

- Fill in the blank at `LINE-1` with the appropriate if statement,
- Fill in the blank at `LINE-2 and LINE-3` with the appropriate return statements.

The program must satisfy the sample input and output. *Marks: 3*

```
#include<iostream>
using namespace std;
bool IsLonger(string str1, string str2){
    if(___________) //LINE-1
        ________________; //LINE-2
    else
        ________________; //LINE-3
}
int main(){
    string str1, str2;
    cin >> str1 >> str2;
    cout << str1 << ", " << str2 << " : " << IsLonger(str1, str2);
    return 0;
}
```

Public 1

```
Input: apple banana
Output: apple, banana : 0
```

Public 2

```
Input: notebook note
Output: notebook, note : 1
```

Private

```
Input: open close
Output: open, close : 0
```

Answer:

```
LINE-1:  str1.length() > str2.length()
LINE-2:  return true
LINE-3:  return false
```

Explanation:

At `LINE-1` the condition must be `if(str1.length() > str2.length())`, then at `LINE-2` it will be `return true;`, otherwise it will be `return false;` at `LINE-3`. This ensures the function returns true if the first string is longer than the second, and false otherwise.

Question 3

Consider the program below.

- Fill in the blank at `LINE-1` to include the appropriate header file to utilize the `abs()` function.
- Fill in the blank at `LINE-2` to compute the Manhattan distance between two points `pt1` and `pt2` as $|pt1.y - pt2.y| + |pt1.x - pt2.x|$.

The program must satisfy the given test cases. *Marks: 3*

```
#include <iostream>
__________________     //LINE-1
using namespace std;
struct Point{
    int x, y;
};

double calculate_distance(Point pt1, Point pt2){
    return ______________________;    //LINE-2
}

int main() {
    int x1, y1, x2, y2;
    cin >> x1 >> y1 >> x2 >> y2;
    Point pt1, pt2;
    pt1.x = x1;
    pt1.y = y1;
    pt2.x = x2;
    pt2.y = y2;
    cout << calculate_distance(pt1, pt2);
    return 0;
}
```

Public 1

```
Input: 2 5 3 8
Output: 4
```

Public 2

```
Input: 10 20 40 20
Output: 30
```

Private

```
Input: 20 40 60 10
Output: 70
```

Answer:

```
LINE-1:  #include <cmath>
LINE-2:  abs(pt1.x - pt2.x) + abs(pt1.y - pt2.y)
```

Explanation:

The C library `math.h` can be included in a C++ program as `#include <cmath>`.

At `LINE-2`, the formula to compute the Manhattan distance between two points can be implemented as:

`abs(pt1.x - pt2.x) + abs(pt1.y - pt2.y)`.

Programming in Modern C++: Assignment Week 2

Total Marks : 25

Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur, Kharagpur – 721302
partha.p.das@gmail.com

July 15, 2024

Question 1

Consider the following code segment. *[MCQ, Marks 2]*

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

int main(){
    int n = 5;
    int *const p = &n;
    int m = 10;
    p = &m; //LINE-1
    cout << *p;
    return 0;
}
```

What will be the output/error?

a) `5`

b) `10`

c) `0`

d) `Compilation Error at LINE-1: assignment of read only variable ‘p’`

Answer: d)
Explanation:
The pointer variable `p` is declared as a constant pointer to an integer, which means the address stored in `p` cannot be changed. At `LINE-1`, we attempt to change the address stored in the constant pointer `p`, leading to a compilation error.

Question 2

Consider the following code segment. *[MCQ, Mark 2]*

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

void update(int &x){
    x += 5;
}

int main(){
    int a = 3;
    int b = 4;
    update(a);
    update(b);
    cout << a << " " << b;
    return 0;
}
```

What will be the output?

a) `3 4`

b) `8 9`

c) `8 4`

d) `3 9`

Answer: b)
Explanation:
The function `update(int &x)` takes an integer reference as a parameter and increments the value of the referenced variable by 5. When `update(a)` is called, the value of `a` becomes 8 (since $3 + 5 = 8$). Similarly, when `update(b)`

Question 3

Consider the following code segment. *[MSQ, Marks 2]*

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

void update(const int &x){
    x = 10; // LINE-1
}

int main(){
    const int a = 5;
    int b = 15;
    const int *p = &a;
    *p = b; // LINE-2
    p = &b; // LINE-3
    update(a); // LINE-4
    update(b); // LINE-5
    return 0;
}
```

Which line/s will give you an error?

a) `LINE-1`

b) `LINE-2`

c) `LINE-3`

d) `LINE-4`

e) `LINE-5`

Answer: a), b)
Explanation:
The function parameter `const int &x` is a reference to a constant integer, so attempting to modify it at `LINE-1` results in a compilation error. The pointer `p` is a pointer to a constant integer, so attempting to modify the value at the location it points to at `LINE-2` results in a compilation error. At `LINE-3`, the pointer `p` is reassigned to point to another address, which is allowed. At `LINE-4`, the constant integer `a` is passed to the function, but since no modification is attempted in the function body, it doesn't cause an error here. At `LINE-5`, the non-const integer `b` is passed to the function, which also doesn't cause an error since it's a valid operation.

Question 4

Consider the following code segment. *[MCQ, Marks 2]*

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

void increment(const int &x) {
    x++; // LINE-1
}

int main() {
    const int a = 10;
    int b = 20;
    increment(a); // LINE-2
    increment(b); // LINE-3
    cout << a << " " << b;
    return 0;
}
```

What will be the output/error?

a) `10 21`

b) `10 20`

c) `Compilation Error:  attempt to increment a constant reference`

d) `Compilation Error:  invalid initialization of non-const reference`

Answer: c)
Explanation:
The function parameter `const int &x` is a reference to a constant integer, so attempting to modify it at `LINE-1` results in a compilation error. Hence, the code will not compile.

Question 5

Consider the following code segment. *[MCQ, Marks 2]*

```
#include <iostream>
using namespace std;
int& modify(int& a) { //LINE-1
    return a = a * 2;
}
int main() {
    int p = 3, q = 4;
    int& r = modify(p);
    cout << p << " " << r << " ";
    modify(p) = q;
    cout << p << " " << r;
    return 0;
}
```

What will be the output?

a) `6 6 4 4`

b) `6 6 8 8`

c) `3 3 4 4`

d) `3 3 8 8`

Answer: a)
Explanation:
The modification of the formal parameter `a` is reflected on the actual variable `p` because it is passed as a reference. The function modifies the value of `p` to `p * 2`. So, the first cout statement will print `6 6`. The statement `int& r = modify(p);` modifies `p` and returns it by reference. The statement `modify(p) = q;` then modifies the value of `p` and `r` to the value of `q`. Hence, the output is `6 6 4 4`.

Question 6

Consider the following code segment. *[MCQ, Marks 2]*

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

int main(){
    int *ptr = ____________________; //LINE-1
    cout << *ptr;
    free(ptr);
    return 0;
}
```

Fill in the blank at `LINE-1` such that the output is `20`.

a) `(int)malloc(20sizeof(int))`

b) `new int`

c) `new int(20)`

d) `new int[20]`

Answer: c)
Explanation:
The pointer variable `ptr` should be assigned with integer type memory and should be initialized with the value 20 to get the desired output. This can be done by `new int(20)`. Hence, the correct option is c).

Question 7

Consider the following code segment. *[MCQ, Marks 2]*

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

enum Days {Sunday, Monday=2, Tuesday, Wednesday, Thursday, Friday, Saturday};

Days operator+(const Days &a, const Days &b){
    unsigned int ea = a, eb = b;
    unsigned int c = (ea + eb) % 7;
    return (Days)c;
}

int main(){
    Days x = Tuesday, y = Friday;
    Days result = x + y;
    cout << result;
    return 0;
}
```

What will be the output/error?

a) `0`

b) `5`

c) `2`

d) `Compilation Error: invalid value in enum`

Answer: c)
Explanation:
The corresponding integer values for `Tuesday` and `Friday` are 3 and 5, respectively. Hence, the addition operator overload will be calculated as `(3 + 5) % 7 = 8 % 7 = 1`, which maps to the second element in the enum, which is `2` (Monday). Therefore, the correct option is c).

Question 8

Consider the following code segment. *[MCQ, Marks 2]*

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

float calculate(int x, float y){
    return x + y;
}
float calculate(float x, int y){
    return x - y;
}

int main(){
    cout << calculate(4, 5); //LINE-1
    return 0;
}
```

What will be the output/error?

a) `9.0`

b) `-1.0`

c) `-1`

d) `Compilation Error at LINE-1: Call of overloaded 'calculate(int, int)' is ambiguous`

Answer: d)
Explanation:
The call to the function `calculate(.)` is ambiguous because it matches both the function prototypes `calculate(int, float)` and `calculate(float, int)`. Hence, the correct option is d). Note that the int data type is implicitly converted to the float data type by the compiler, which leads to the ambiguity.

Question 9

Consider the following code segment. *[MSQ, Marks 2]*

```
#include<iostream>
using namespace std;
struct vector{
    int x, y;
    void show(){ cout << "(" << x << ", " << y << ")"; }
};
--------------------------------------{ //LINE-1
    vector temp;
    temp.x = v1.x * v2.x;
    temp.y = v1.y * v2.y;
    return temp;
}
int main(){
    struct vector v1={2,3},v2={4,5};
    struct vector t = v1 * v2;
    t.show();
    return 0;
}
```

Fill in the blank at `LINE-1` with the correct function header.

a) `vector operator*(vector &v1, vector &v2)`

b) `vector operator*(vector v1, vector v2)`

c) `int operator*(vector v1, vector v2)`

d) `void operator*(vector v1, vector v2)`

Answer: a), b)
Explanation: We need to overload the multiplication operator for the structure vector. Both options a and b are correct, as they take two vector objects as parameters and return a new vector object that is the result of multiplying the two input vectors. Option a passes the parameters by reference, while option b passes them by value. Both approaches are valid.

Programming Questions

Question 1

Consider the program below.

- Fill in the blank at `LINE-1` and `LINE-2` with appropriate statements

such that it satisfies the given test cases. *Marks: 3*

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

float compute(__________ x){ //LINE-1
    return (____); //LINE-2
}

int main(){
    float y;
    cin >> y;
    cout << compute(y * 2);
    return 0;
}
```

Public 1

```
Input: 3.5
Output: 6
```

Public 2

```
Input: 2.0
Output: 3
```

Private 3

```
Input: 5.5
Output: 10
```

Answer:
`LINE-1: const float&`
`LINE-2: x - 1`
Explanation:
Since we are passing an expression as an argument, the header for the function `compute()` will be: `float compute(const float& x)`.
Since `x` is a constant, it cannot be changed by decrement operator. But we need to decrement the argument by 1 to get the result. Hence, `LINE-2` will be filled as `return (x - 1)`.

Question 2

Consider the following program that is intended to reverse a given string and check if it is a palindrome.

- Fill in the blanks at `LINE-1` and `LINE-2` to complete the function definitions

such that it satisfies the given test cases. *Marks: 3*

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

string reverseString(const string& str) {
    ______________ ;// LINE-1
    for (int i = str.length() - 1; i >= 0; i--) {
        reversed += str[i];
    }
    return reversed;
}

bool isPalindrome(const string& str) {
    ______________ ;// LINE-2
    return str == reversedStr;
}

int main(){
    string input;
    cin >> input;
    string reversedInput = reverseString(input);
    cout << "Reversed: " << reversedInput << endl;
    if (isPalindrome(input)) {
        cout << "The string is a palindrome." << endl;
    } else {
        cout << "The string is not a palindrome." << endl;
    }
    return 0;
}
```

Public 1

```
Input: level
Output:
Reversed: level
The string is a palindrome.
```

Public 2

```
Input: hello
Output:
Reversed: olleh
The string is not a palindrome.
```

Private

```
Input: radar
Output:
Reversed: radar
The string is a palindrome.
```

Answer

```
LINE-1:  string reversed = "";
LINE-2:  string reversedStr = reverseString(str);
```

Explanation:

In LINE-1, to reverse the string correctly, we initialize an empty string `reversed` and append characters from the input string in reverse order.
In LINE-2, for the palindrome check, we call the `reverseString` function and compare the original string with the reversed string. If they are equal, the original string is a palindrome.

Question 3

Consider the following program.

- Fill in the blank at `LINE-1` to complete the operator function header for the addition operator +.
- Fill in the blank at `LINE-2` with a proper statement

such that the program must satisfy the sample input and output. *Marks: 3*

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

struct Vector{
    int x;
    int y;
};

_____________________________(const Vector& v1, const Vector& v2){ //LINE-1
    Vector v;
    v.x = v1.x + v2.x;
    v.y = v1.y + v2.y;
    return v;
}

Vector wrapper(const Vector v1, const Vector v2){
    Vector v = _________; //LINE-2
    return v;
}

int main(){
    int a, b, c, d;
    cin >> a >> b >> c >> d;
    Vector v1 = {a, b}, v2 = {c, d};
    Vector result = wrapper(v1, v2);
    cout << result.x << " " << result.y;
    return 0;
}
```

Public 1

```
Input: 1 2 3 4
Output:
4 6
```

Public 2

```
Input: 5 5 10 10
Output: 15 15
```

Private

```
Input: -1 -2 -3 -4
Output: -4 -6
```

Answer:

`LINE-1: Vector operator+`

`LINE-2: v1 + v2`

Explanation:

The operator function calculates vector addition. So, this addition operator needs to be overloaded for the Vector structure. Hence, `LINE-1` should be filled as

`Vector operator+(const Vector& v1, const Vector& v2)`

To call the operator function, `LINE-2` should be filled as

`v = v1 + v2`.

Programming in Modern C++: Assignment Week 3

Total Marks : 25

Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur, Kharagpur – 721302
partha.p.das@gmail.com

July 31, 2024

Question 1

Consider the following code segment. *[MCQ, Marks 2]*

```
class Rectangle{
    private:
        int width, height;
    public:
        void setWidth(int w){ width = w; }
        void setHeight(int h){ height = h; }
        void display(){ cout << width << " x " << height; }
        void doubleSize(){ width *= 2; height *= 2; }
        int increaseWidth(){ return width + 1; }
        int increaseHeight(){ return ++height; }
};
```

Identify the set of all the methods that change the state of the `class Rectangle` objects.

a) `setWidth(), setHeight(), display()`

b) `setWidth(), setHeight(), doubleSize(), increaseHeight()`

c) `setWidth(), setHeight(), doubleSize(), increaseWidth(), increaseHeight()`

d) `setWidth(), setHeight(), increaseWidth(), increaseHeight()`

Answer: b)
Explanation:
The function `setWidth()` changes the data member value `width`. Thus, `setWidth()` changes the state of the object.
The function `setHeight()` changes the data member value `height`. Thus, `setHeight()` changes the state of the object.
The function `doubleSize()` changes the values of data members `width` and `height`. Thus, `doubleSize()` changes the state of the object.
The function `increaseHeight()` changes the value of data member `height`. Thus, `increaseHeight()` changes the state of the object.
Note that, the function `increaseWidth()` does not change the value of data member `width`, as it returns an expression only i.e. `(width + 1)`.

Question 2

Consider the following code segment. *[MCQ, Marks 2]*

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

class Example {
    int x;
    public:
        Example(int x_ = 0) : x(x_) { cout << "Constructor:" << x << " "; }
        ~Example() { cout << "Destructor:" << x << " "; }
};

Example globalObj(50); // LINE-1

int main() {
    Example *pExample = new Example(20); // LINE-2
    Example localObj1(30); // LINE-3
    {
        Example localObj2(40); // LINE-4
        delete pExample; // LINE-5
    } // LINE-6
    return 0;
} // LINE-7
```

What will be the output?

a) `Constructor:50 Constructor:20 Constructor:30 Constructor:40 Destructor:40 Destructor:30 Destructor:20 Destructor:50`

b) `Constructor:20 Constructor:50 Constructor:30 Constructor:40 Destructor:40 Destructor:30 Destructor:50 Destructor:20`

c) `Constructor:50 Constructor:20 Constructor:40 Constructor:30 Destructor:30 Destructor:40 Destructor:20 Destructor:50`

d) `Constructor:50 Constructor:20 Constructor:30 Constructor:40 Destructor:20 Destructor:40 Destructor:30 Destructor:50`

Answer: d)
Explanation:
The object (`globalObj`) defined at `LINE-1` has a global scope, so this object will be created even before the `main()` function starts. It calls the constructor and prints `Constructor: 50`. Then, within `main()` another object gets instantiated at `LINE-2`, and it calls the constructor and prints `Constructor: 20`.
At `LINE-3`, an object `localObj1` gets instantiated, which calls the constructor and prints `Constructor: 30`.
At `LINE-4`, an object `localObj2` gets instantiated within a block scope, and calls the constructor and it prints `Constructor: 40`.
At `LINE-5`, the object `pExample` (which was created at `LINE-2`) gets deleted, so it calls the destructor and prints `Destructor: 20`.
At `LINE-6`, the block scope ends. Thus, the local object `localObj2` (which was created at `LINE-4`) gets deleted. It calls the destructor and prints `Destructor: 40`.

At `LINE-7`, the scope of the `main()` function ends. Thus, the local object `localObj1` (which was created at `LINE-3`) gets deleted. It calls the destructor and prints `Destructor:  30`.
At the end of the program, the object `globalObj` having a global scope gets deleted. It calls the destructor and prints `Destructor:  50`.

Question 3

Consider the following class. *[MCQ, Mark 1]*

```
class Sample {
    ________:
        int a;
    ________:
        int b;
    /* Some more code */
};
```

Fill in the blanks with proper access specifiers so that member `b` can be accessed from outside of the class but member `a` cannot be accessed.

a) `public, public`

b) `private, public`

c) `public, private`

d) `private, private`

Answer: b)
Explanation:
A class member should be declared as `public` to be accessed from outside of the class. On the other hand, a private data member prevents itself from being accessed directly from outside of the class. Therefore, member `a` should be private and member `b` should be public.

Question 4

Consider the code segment below. *[MCQ, Marks 2]*

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

int globalVar = 5;

class MyClass {
    int memberVar;
    public:
        MyClass(int memberVar_ = 0) : memberVar(++memberVar_) { ++globalVar; } // LINE-1
        ~MyClass() { memberVar = 0; globalVar = 0; }
    void display() {
        cout << "memberVar = " << memberVar << ", globalVar = " << globalVar << endl;
    }
};

void function() {
    MyClass obj;
    obj.display();
}

int main() {
    MyClass obj;
    function();
    obj.display();
    return 0;

}
```

What will be the output?

a) `memberVar = 6, globalVar = 7`
   `memberVar = 1, globalVar = 8`

b) `memberVar = 1, globalVar = 7`
   `memberVar = 1, globalVar = 0`

c) `memberVar = 1, globalVar = 8`
   `memberVar = 1, globalVar = 0`

d) `memberVar = 1, globalVar = 7`
   `memberVar = 2, globalVar = 0`

Answer: b)
Explanation:
The statement `MyClass obj;` in `main()` calls the constructor at `LINE-1`, which makes `memberVar = 1` and `globalVar = 6`.
Then the statement `MyClass obj;` in `function()` calls the constructor at `LINE-1`, which makes `memberVar = 1` and `globalVar = 7`.
Then the statement `obj.display();` in `function()` prints `memberVar = 1`, `globalVar = 7`.
As the function `function()` returns, the destructor for the local object `obj` is called, which makes `memberVar = 0` and `globalVar = 0`.
Finally, the statement `obj.display();` in `main()` prints `memberVar = 1`, `globalVar = 0`.

Question 5

Consider the following code segment. *[MCQ, Marks 1]*

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

int globalVar = 0;

class Example {
    public:
        Example() { globalVar = 5; }
        ~Example() { globalVar = 6; }
};

void someFunction() {
    Example ex;
}

int anotherFunction() {
    globalVar = 7;
    someFunction();
    return globalVar++;
}

int main() {
    cout << anotherFunction() << " ";
    cout << globalVar << endl;
    return 0;
}
```

What will be the output?

a) `5 6`

b) `7 5`

c) `6 7`

d) `7 8`

Answer: c)
Explanation:
`globalVar` is initialized to 0 (`globalVar = 0;` executes before `main()` is called). Then `main()` starts and calls `anotherFunction()` which sets `globalVar` to 7 by `globalVar = 7;`. The `anotherFunction()` calls the function `someFunction()`. In function `someFunction()`, `Example ex` sets `globalVar = 5`. But the object `ex` is local within function `someFunction()`. So, as the function `someFunction()` returns, the destructor of local object `ex` will be called before return and `globalVar` becomes 6. This 6 will be returned by `anotherFunction()`, and gets printed.
Further, `globalVar` value is incremented to 7 after return (since `globalVar++` is post-increment). Finally, the value of `globalVar` is printed as 7.
Hence the correct option is c).

Question 6

Consider the code segment below. *[MCQ, Marks 2]*

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

class Sample {
    int x, y, z;
    public:
        Sample(int val = 0) : z(++val), y(val++), x(++val) {}
    void display() {
        cout << "x = " << x << ", y = " << y << ", z = " << z << endl;
    }
};

int main() {
    Sample obj(5);
    obj.display();
    return 0;
}
```

What will be the output?

a) `x = 8, y = 6, z = 6`

b) `x = 6, y = 6, z = 8`

c) `x = 8, y = 7, z = 6`

d) `x = 6, y = 7, z = 8`

Answer: b)
Explanation:
The order in which the data members are initialized in an initialization list follows the order of declaration of the data members.
Since `z` is declared first, `z` gets initialized to `++val` i.e. 6.
Then `y` gets initialized to `val++` i.e. 6. However, the value of `val` becomes 7.
Finally, `x` gets initialized to `++val` i.e. 8.

Question 7

Consider the code segment below. *[MCQ, Marks 2]*

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

class Car {
    int id;
    string model;
    Car(){} // LINE-1
    public:
        Car(int id_, string model_) : id(id_), model(model_) {}
    void updateCar(Car& c) {
        this->id = c.id;
        this->model = c.model;
    } // LINE-2
    void display() {
        cout << id << ", " << model << endl;
    }
};

int main() {
    Car car1(101, "Sedan");
    Car car2(202, "SUV");
    car1.updateCar(car2);
    car1.display();
    return 0;
}
```

What will be the output/error?

a) `101, Sedan`

b) `202, SUV`

c) `Compiler error at LINE-1: Car::Car() cannot be private`

d) `Compiler error at LINE-2: lvalue required as left operand of assignment`

Answer: d)
Explanation:
Since `this` is a constant pointer, any attempt to modify it (at `LINE-2`) results in a compiler error.

Question 8

Consider the code segment below. *[MCQ, Marks 2]*

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

class Rectangle {
    int width, height;
    public:
        Rectangle(int width_ = 0, int height_ = 0) : width(width_), height(height_) {
        cout << "ctor ";
    } // LINE-1

    Rectangle(Rectangle &r) : width(r.width), height(r.height) {
        cout << "c-ctor ";
    } // LINE-2

    Rectangle& operator=(Rectangle r) {
        width = r.width;
        height = r.height;
        cout << "c-assign ";
        return *this;
    } // LINE-3
};

int main() {
    Rectangle r1(30, 40);
    Rectangle r2 = r1;
    Rectangle *rPtr;
    Rectangle r3;
    r3 = r2;
    return 0;
}
```

What will be the output/error?

a) `ctor c-ctor ctor c-ctor`

b) `ctor c-ctor ctor c-assign`

c) `ctor c-ctor ctor c-ctor c-assign`

d) `ctor c-assign ctor c-assign`

Answer: c)
Explanation:
The statement `Rectangle r1(30, 40);` invokes the constructor at `LINE-1` and prints `ctor`.
The statement `Rectangle r2 = r1;` invokes the copy constructor at `LINE-2` and prints `c-ctor`.
The statement `Rectangle *rPtr;` does not create an object.
The statement `Rectangle r3;` invokes the constructor at `LINE-1` and prints `ctor`.
The statement `r3 = r2;` calls the copy assignment function. Since it passes `r2` as pass by value, it invokes the copy constructor at `LINE-2` and prints `c-ctor`. Then it executes the body of the copy assignment function (at `LINE-3`) and prints `c-assign`.

Question 9

Consider the following code segment. *[MCQ, Marks 2]*

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

class Demo {
    int y;
    public:
        Demo(int b=0) : y(b){ cout << "First "; }
        Demo(const int &j) : y(j){ cout << "Second "; }
};

int main() {
    Demo d1 = 10;
    return 0;
}
```

What will be the output/error?

a) `First`

b) `Second`

c) `First Second`

d) `Compilation error:  conversion from 'int' to 'Demo' is ambiguous`

Answer: d)
Explanation:
Both the constructors take an integer as input and assign it to the data member of the class. So, when the assignment of an integer to the class object is done, the compiler becomes confused about which constructor to call. Hence, it gives an error.

Programming Questions

Question 1

Marks: 3

Consider the following program. Fill in the blanks as per the instructions given below:

- at `LINE-1` with an appropriate initialization block to initialize the data members,
- at `LINE-2` with an appropriate definition of the destructor to free the memory allocated for the data members,
- at `LINE-3` and `LINE-4` with appropriate definitions of the functions `getA()` and `getB()`.

such that it will satisfy the given test cases.

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

class Pair {
    const int *a, *b;
    public:
        Pair(int val1, int val2) : ____________________________{}    //LINE-1
        ~Pair(){ ___________________________________________ }    //LINE-2
        int getA(){ ________________________________________ }    //LINE-3
        int getB(){ ________________________________________ }    //LINE-4
};

int main(){
    int x, y;
    cin >> x >> y;
    Pair p(x, y);
    cout << "[" << p.getA() << ", " << p.getB() << "]";
    return 0;
}
```

Public 1

```
Input: 3 4
Output: [3, 4]
```

Public 2

```
Input: 7 8
Output: [7, 8]
```

Private

```
Input: 15 25
Output: [15, 25]
```

Answer:

```
LINE-1:  a(new int(val1)), b(new int(val2))
LINE-2:  delete a; delete b;
LINE-3:  return *a;
```

```
LINE-4:  return *b;
```

Explanation:

Since the data members are pointer types, the constructor must allocate memory and initialize them. Thus, the definition of the constructor can be as follows:

```
Pair(int val1, int val2) :  a(new int(val1)), b(new int(val2))  //LINE-1
```

The destructor must free the memory allocated for the data members. Thus, the definition of the destructor can be as follows:

```
Pair() delete a; delete b;  //LINE-2
```

The functions `getA()` and `getB()` must return the values pointed to by the data members `a` and `b` respectively. Thus, the definitions of the functions can be as follows:

```
int getA() return *a;  //LINE-3
int getB() return *b;  //LINE-4
```

Question 2

. *Marks: 3*

Consider the following program. Fill in the blanks as per the instructions given below:

- at `LINE-1` with appropriate declaration of the data member `total`,
- at `LINE-2` and `LINE-3` with appropriate headers for the functions `calculateTotal()` and `display()`

such that it will satisfy the given test cases.

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

class Calculator {
    int a, b;
    ________________;                                          //LINE-1
    public:
        Calculator(int a_, int b_) : a(a_ * 2), b(b_ * 2){}
        ___________________________________ { total = a + b; };       //LINE-2
        ___________________________________ {                      //LINE-3
            cout << "a = " << a << ", b = " << b << ", total = " << total;
        }
};

int main(){
    int x, y;
    cin >> x >> y;
    const Calculator calc(x, y);
    calc.calculateTotal();
    calc.display();
    return 0;
}
```

Public 1

```
Input: 4 6
Output: a = 8, b = 12, total = 20
```

Public 2

```
Input: 7 -3
Output: a = 14, b = -6, total = 8
```

Private

```
Input: 15 10
Output: a = 30, b = 20, total = 50
```

Answer:

```
LINE-1:  mutable int total
LINE-2:  void calculateTotal() const
LINE-3:  void display() const
```

Explanation:

Since `calc` is defined as a constant object, and we need to modify the value of `total`, `total` has to be defined as a `mutable` member. Thus, the declaration of `total` can be as follows:

```
LINE-1:  mutable int total
```

Since the functions `calculateTotal()` and `display()` are called on a constant object, they must be defined as constant functions as follows:

```
LINE-2:  void calculateTotal() const
LINE-3:  void display() const
```

Question 3

Marks: 3

Consider the following program. Fill in the blanks as per the instructions given below:

- at `LINE-1` with the appropriate definition of the copy constructor,
- at `LINE-2` with the appropriate definition of the copy assignment operator,
- at `LINE-3` with the appropriate condition to check for self-assignment.

such that it will satisfy the given test cases.

```
#include<iostream>
#include<cstring>
#include<cstdlib>
using namespace std;

class Student {
    int sid;
    char *name;
    public:
        Student(int sid_, const char *name_) : sid(sid_), name(strdup(name_)) {}
        ______________________ : sid(s.sid), name(strdup(s.name)) { }   //LINE-1
        ____________________________________ {    //LINE-2
            if (_____________________) {             //LINE-3
                free(name);
                sid = s.sid;
                name = strdup(s.name);
            }
            return *this;
        }
        void display(){
             cout << sid << " : " << name << endl;
        }
};

int main(){
    int a, b;
    char n1[80], n2[80];
    cin >> a >> n1 >> b >> n2;
    Student s1(a, n1);
    Student s2 = s1;
    Student s3(b, n2);
    s1 = s3;
    s1.display();
    s2.display();
    s3.display();
    return 0;
}
```

Public 1

```
Input:
5 John
10 Alice
Output:
10 : Alice
5 : John
10 : Alice
```

Public 2

```
Input:
20 Bob
25 Carol
Output:
25 : Carol
20 : Bob
25 : Carol
```

Private

```
Input:
30 Dave
35 Eve
Output:
35 : Eve
30 : Dave
35 : Eve
```

Answer:

```
LINE-1:  Student(const Student& s)
LINE-2:  Student& operator=(const Student& s)
LINE-3:  this != &s
```

Explanation:

At `LINE-1`, the header of the copy constructor must be as follows: `LINE-1:  Student(const Student& s)`

At `LINE-2`, the header of the copy assignment operator must be as follows: `LINE-2:  Student& operator=(const Student& s)`

At `LINE-3`, the condition must check for self-assignment which can be done using the code as follows:

```
LINE-3:  this != &s
```

Programming in Modern C++: Assignment Week 4

Total Marks : 25

Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur, Kharagpur – 721302
partha.p.das@gmail.com

August 6, 2024

Question 1

Consider the following code segment. *[MCQ, Marks 2]*

```
#include <iostream>

using namespace std;

class Counter {
    static int count;

public:
    void increment() {
        count = count + 5;
    }

    void display() {
        cout << count;
    }
};

int Counter::count = 15;

int main() {
    Counter c1, c2;
    c1.increment();
    c2.increment();
    c2.display();
    return 0;
}
```

What will be the output?

a) `15`

b) `20`

c) `25`

d) `30`

Answer: c)
Explanation:
The `static` data member `Counter::count` is initialized during the program startup and is shared by both objects `c1` and `c2` of the class `Counter`. Hence, the data member is initialized as 15 during the start of the `main` function. It is incremented two times, each time by 5, when the `increment()` function is called for both objects. Hence, the `display()` function will print 25.

Question 2

Consider the following code segment. *[MCQ, Marks 2]*

```
#include <iostream>

using namespace std;

class Example {
    static int count;

public:
    void increment() {
        count = count + 5;
        show();
    }

    ______ void show() {  // LINE-1
        cout << count;
    }
};

int Example::count = 0;

int main() {
    Example e;
    e.increment();
    return 0;
}
```

Fill in the blank at `LINE-1` such that the program will print `5`.

a) `mutable`

b) `static`

c) `class`

d) `friend`

Answer: b)
Explanation:
The `show()` function is being called from the `increment()` function without any object. It can be done only when the function is declared as `static`. Hence, `LINE-1` should be filled as `static`.

Question 3

Consider the following code segment. *[MSQ, Marks 2]*

```
#include <iostream>

using namespace std;

class Number {
    int value;

public:
    Number(int _value = 0) : value(_value) {}

    ______________________________  // LINE-1
};

void show(Number &n) {
    cout << n.value;
}

int main() {
    Number n(10);
    show(n);
    return 0;
}
```

Fill in the blank at `LINE-1` such that the program will print 10.

a) `void friend show(Number&);`

b) `static void show(Number&);`

c) `friend void show(Number&);`

d) `void const show(Number&);`

Answer: a), c)
Explanation:
The global function `show()` accesses the private data member of the class `Number`. This can only be done when the function is declared as a `friend` of the class. Hence, the correct options are a) and c).

Question 4

Consider the following code segment. *[MCQ, Marks 2]*

```
#include <iostream>

using namespace std;

int num = 5;

namespace space {
    int num = 15;
}

int main() {
    using namespace space;

    int num = 10;

    {
        cout << ::num << " " << num << " " << space::num;
    }

    return 0;
}
```

What will be the output/error?

a) `5 10 15`

b) `10 5 15`

c) `15 5 10`

d) `Compilation Error: reference to 'num' is ambiguous`

Answer: a)
Explanation:
When there are multiple instances of the same variable, the local instance will get higher priority. So, `num` will be printed as 10. To access the global variable, we use `::num`. For the namespace variable, it is qualified by the namespace `space`. So, the cout statement at `LINE-1` will print `5 10 15`.

Question 5

Consider the following code segment. *[MCQ, Marks 2]*

```
#include <iostream>

int main() {
    char message[] = "Welcome";
    cout << message; // LINE-1
    return 0;
}
```

The `cout` statement at `LINE-1` gives an error. Change the `cout` statement such that it prints `Welcome`.

a) `std::cout << message;`

b) `using cout << message;`

c) `using std::cout << message;`

d) `std.cout << message;`

Answer: a)
Explanation:
`cout` is a predefined object in the namespace `std`. In order to call the `cout` function, we need to specify that `cout` belongs to the `std` namespace. The correct syntax for the same is `std::cout << message;` i.e. option a).

Question 6

Consider the following code segment. *[MCQ, Marks 2]*

```
#include <iostream>

using namespace std;

int y = 2;

namespace name2 {
    int y = 4;
}

int main() {
    int y = 3;
    cout <<_________________________<< endl; // LINE-1
    return 0;
}
```

Fill in the blank at `LINE-1` so that the program will print 6.

a) `name2::y + ::y`

b) `name2::y + y`

c) `y + ::y`

d) `name2.y + ::y`

Answer: a)
Explanation:
As per the output, we need to add the global `y` and `y` from the namespace `name2`. It can be done as `name2::y + ::y`. Hence, the correct option is a).

Question 7

Consider the following code segment. *[MCQ, Marks 2]*

```
#include <iostream>

using namespace std;

class X {
    int x;

public:
    X(int _x) : x(_x) {}

    int getValue() {
        return x;
    }
};

class Y {
    static X obj;

public:
    static int getValue() {
        return obj.getValue();
    }
};

int main(void) {
    cout << Y::getValue();
    return 0;
}
```

What will be the output/error?

a) `0`

b) `10`

c) `Compilation error: cannot access static object obj`

d) `Compilation error: undefined reference Y::obj`

Answer: d)
Explanation:
The static variable `obj` in `Y` needs to be initialized globally in order to use it. But there is no initialization for `obj` in the program. Thus, it gives a compilation error as "undefined reference to `Y::obj`".

Question 8

Consider the following code segment. *[MCQ, Marks 1]*

```
#include <iostream>

using namespace std;

class Counter {
    static int count;

public:
    Counter() {
        count++;
    }

    static int getCount() {
        return count;
    }
};

int Counter::count = 5;

int main() {
    cout << Counter::getCount() << " ";
    Counter c[3];
    cout << Counter::getCount();
    return 0;
}
```

What will be the output?

a) `5 8`

b) `5 9`

c) `6 8`

d) `6 9`

Answer: a)
Explanation:
The lifetime of a static class variable will be throughout the program. So, the static data member of the class is initialized with 5 when the `getCount()` function is called. The first `cout` statement prints 5. Next, the initialization of another three objects increments the static variable three times. Hence, the second `cout` statement prints 8.

Question 9

Consider the following code segment. *[MCQ, Marks 1]*

```
#include <iostream>

using namespace std;

class Alpha {
    int x = 15;

    _________________________;// LINE-1:
};

class Beta {
public:
    int increase(Alpha &a) {
        return (a.x + 5);
    }
};

int main() {
    Alpha a1;
    Beta b1;
    cout << b1.increase(a1);
    return 0;
}
```

Fill in the blank at `LINE-1` such that the program will print `20`.

a) `friend class Beta`

b) `class friend Beta`

c) `friend int increase(Alpha&)`

d) `friend int Beta::increase(Alpha&)`

Answer: a)
Explanation:
The `increase()` function of `Beta` is accessing the private member of `Alpha`. It is possible when the class `Beta` is declared as a friend of class `Alpha`. So, the correct answer is a). The function `increase()` cannot be declared as a friend of class `Alpha` because of forward declaration problems.

Programming Questions

Question 1

Marks: 3

Consider the following program. Fill in the blanks as per the instructions given below:

- at `LINE-1` with appropriate declaration so that a global operator function can access private data of `class Vector`,
- at `LINE-2` with the appropriate operator function header,
- at `LINE-3` with the appropriate return statement, such that it will satisfy the given test cases.

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

class Vector{
    int x, y;
public:
    Vector(int a, int b) : x(a), y(b){}
    void display(){
        cout << x << " " << y;
    }

    ____________________________________________;        //LINE-1
};

___________________________________{                                  //LINE-2
    return Vector(______________);                      //LINE-3
}

int main(){
    int n;
    cin >> n;
    Vector v(2, 3);
    Vector v1 = v + n;
    v1.display();
    return 0;
}
```

Public 1

```
Input: 4
Output: 6 3
```

Public 2

```
Input: 7
Output: 9 3
```

Private

```
Input: 1
Output: 3 3
```

Answer:

```
LINE-1:  friend Vector operator+(const Vector& v, int n)
LINE-2:  Vector operator+(const Vector& v, int n)
LINE-3:  v.x + n, v.y
```

Explanation:

We need to overload the addition operator for the `Vector` class so that a `Vector` object and an integer can be added together and return the resultant `Vector` object.

Hence, `LINE-2` will be filled as `Vector operator+(const Vector& v, int n)`

The operator function should have access to the private data members of the `Vector` class.

Hence, `LINE-1` should be filled as `friend Vector operator+(const Vector& v, int n)`.

As per the test cases, `LINE-3` should be filled as `v.x + n, v.y`.

Question 2

Marks: 3

Consider the following program. Fill in the blanks as per the instructions given below:

- at `LINE-1` with the appropriate keyword to declare the `Singleton` pointer variable,
- at `LINE-2` to complete the header for function `getInstance(int value)`,
- at `LINE-3` with the appropriate keyword to complete `instance` variable initialization such that it will satisfy the given test cases.

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

class Singleton {
    int data;
    ______ Singleton *instance;                          //LINE-1
    Singleton(int value) : data(value) {}
public:
    int getData() { return data; }
    ________________ getInstance(int value) {        //LINE-2
        if(!instance)
            instance = new Singleton(value);
        return instance;
    }
};

___________ Singleton::instance = 0;                  //LINE-3

int main(){
    int n, value;
    cin >> n;
    int arr[n];
    for(int i = 0; i < n; i++)
        cin >> arr[i];
    for(int i = 0; i < n; i++){
        Singleton *instance = Singleton::getInstance(arr[i]);
        cout << instance->getData() << " ";
    }
    return 0;
}
```

Public 1

```
Input:
2
3 4
Output: 3 3
```

Public 2

```
Input:
3
```

```
2 4 6
Output: 2 2 2
```

Private

```
Input:
2
5 7
Output: 5 5
```

Answer:

```
LINE-1:  static
LINE-2:  static Singleton*
LINE-3:  Singleton*
```

Explanation:

As per the test cases given, the pointer variable `instance` is initialized only once at the time of first object creation in the main function. The same instance is used for all objects. It can be done only when the `LINE-1` is filled as `static`.

The function `getInstance()` is being called using the class name. It can only be done when the function is declared static. Hence, `LINE-2` will be filled as `static Singleton*`.

The initialization of `static` variable at `LINE-3` can be done as `Singleton* Singleton::instance = 0;`

Question 3

Marks: 3

Consider the following program. Fill in the blanks as per the instructions given below:

- at `LINE-1` with an appropriate statement so that the global operator function can access private members,
- at `LINE-2` with an appropriate operator function header,
- at `LINE-3` with an appropriate operator function header,
- at `LINE-4` with an appropriate operator function header such that it will satisfy the given test cases.

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

class Vector {
    int x;
    int y;
public:
    Vector(int _x, int _y) : x(_x), y(_y) {}
    void display(){
        cout << "(" << x << "," << y << ")";
    }
    ____________________________________________;        //LINE-1

    ____________________{                                //LINE-2
        ++x;
        return *this;
    }
    ______________________{                              //LINE-3
        Vector temp(x, y);
        ++y;
        return temp;
    }
};
_________________(istream& is, Vector& v){               //LINE-4
    is >> v.x >> v.y;
    return is;
}
int main(){
    int i, j;
    cin >> i >> j;
    Vector v(i, j);
    ++(++v);
    v++;
    v.display();
    return 0;
}
```

Public 1

`Input:`

```
3 4
Output:
(5,5)
```

Public 2

```
Input:
5 2
Output:
(7,3)
```

Private

```
Input:
1 5
Output:
(3,6)
```

Answer:

```
LINE-1:  friend istream& operator>>(istream&, Vector&)
LINE-2:  Vector& operator++()
LINE-3:  Vector operator++(int)
LINE-4:  istream& operator>>
```

Explanation:

The input operator needs to be overloaded for class `Vector` to take data members as input. Hence, `LINE-4` can be filled as `istream& operator>>`.

As per the test cases, the pre-increment operator for the class object `v` is called twice, which increments the data member `x` keeping the data member `y` as it is. Hence, `LINE-2` should be filled as `Vector& operator++()`.

Similarly, for the post-increment operator, `LINE-3` should be filled as `Vector operator++(int)`.

The global operator function at `LINE-4` should have access to the private data members of class `Vector`. Hence, `LINE-1` should be filled as `friend istream& operator>>(istream&, Vector&)`.

Programming in Modern C++: Assignment Week 5

Total Marks : 25

Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur, Kharagpur – 721302
partha.p.das@gmail.com

August 13, 2024

Question 1

Consider the following code segment. *[MCQ, Marks 1]*

```
#include <iostream>

using namespace std;

class Base {
protected:
    int number;
    static int counter;

public:
    Base() {
        number = ++counter;
        cout << number << "-" << counter << ", ";
    }
};

class Derived1 : public Base {
public:
    Derived1() {
        ++counter;
        cout << counter << ", ";
    }
};

class Derived2 : public Derived1 {
public:
    Derived2() {
        ++counter;
        cout << counter << ", ";
    }
};

int Base::counter = 0;
```

```
int main() {
    Derived2 dObj[2];
    return 0;
}
```

What will be the output?

a) `1, 2, 3`

b) `1, 2, 3-3, 1, 2, 3-3,`

c) `1, 2, 3-3, 4, 5, 6-6,`

d) `1-1, 2, 3, 4-4, 5, 6,`

Answer: d)
Explanation:
When we create an object, all the constructors of the class hierarchy get executed in top-down order. Since we create 2 objects of `Derived2` type. For the first object, `class Base` prints `1-1`, `class Derived1` prints `2` and `class Derived2` prints `3`. For the second object, `class Base` prints `4-4`, `class Derived1` prints `5` and `class Derived2` prints `6`.

Question 2

Consider the following code segment. *[MCQ, Marks 1]*

```
#include <iostream>

using namespace std;

class Alpha {
public:
    Alpha() {
        cout << "A ";
    }

    ~Alpha() {
        cout << "-A ";
    }
};

class Beta : public Alpha {
public:
    Beta() {
        cout << "B ";
    }

    ~Beta() {
        cout << "-B ";
    }
};

class Gamma : public Alpha {
    Beta b;

public:
    Gamma() {
        cout << "C ";
    }

    ~Gamma() {
        cout << "-C ";
    }
};

int main() {
    Gamma g1;
    return 0;
}
```

What will be the output?

a) `A B C -C -B -A`

b) `A A B C -C -A -A`

c) `A C -C -A`

d) `A A B C -C -B -A -A`

Answer: d)
Explanation:
When an object of class `Gamma` is being instantiated, the constructor of class `Alpha` is called, which will print "A" first. Then the data member of class `Gamma` is created, which will again call the constructor of class `Alpha` and print "A", then print "B" from the constructor of class `Beta`. Finally, "C" is printed from the constructor of class `Gamma`. After the end of the `main()` function, the reverse of the already printed sequence will be printed from the destructors of the classes. So, the answer is (d).

Question 3

Consider the following code segment. *[MCQ, Marks 2]*

```
#include <iostream>

using namespace std;

class Vehicle {
public:
    Vehicle() {
        cout << "Vehicle created, ";
    }

    ~Vehicle() {
        cout << "Vehicle destroyed, ";
    }
};

class Car : public Vehicle {
public:
    Car() {
        cout << "Car created, ";
    }

    ~Car() {
        cout << "Car destroyed, ";
    }
};

class Bike : public Vehicle {
public:
    Bike() {
        cout << "Bike created, ";
    }

    ~Bike() {
        cout << "Bike destroyed, ";
    }
};

class SportsCar : public Car {
    Bike b;

public:
    SportsCar() {
        cout << "SportsCar created, ";
    }

    ~SportsCar() {
        cout << "SportsCar destroyed, ";
    }
};
```

```
int main() {
    SportsCar sc;
    return 0;
}
```

What will be the output?

a) `Vehicle created, Car created, Vehicle created, Bike created, SportsCar created, SportsCar destroyed, Bike destroyed, Vehicle destroyed, Car destroyed, Vehicle destroyed,`

b) `Vehicle created, Car created, SportsCar created, Vehicle created, Bike created, Bike destroyed, Vehicle destroyed, SportsCar destroyed, Car destroyed, Vehicle destroyed,`

c) `Vehicle created, Car created, SportsCar created, Bike created, Bike destroyed, SportsCar destroyed, Car destroyed, Vehicle destroyed,`

d) `Vehicle created, Car created, SportsCar created, Bike created, Vehicle destroyed, Car destroyed, SportsCar destroyed, Bike destroyed,`

Answer: a)
Explanation:
A constructor of the derived class must first call the constructors of the base classes to construct the base class instances of the derived class.
The destructor of the derived class must call the destructor of the base classes to destruct the base class instances of the derived class.
Thus, to instantiate an object of `class SportsCar`, it first calls the constructor of `class Vehicle` which prints `Vehicle created`. Then the constructor of `class Car` which prints `Car created`. However, to create an object of `SportsCar`, an object of `class Bike` needs to be instantiated. The creation of the `Bike` object first calls the constructor of `class Vehicle` which prints `Vehicle created`, and then the constructor of `class Bike` which prints `Bike created`. Finally, the constructor of `SportsCar` is called which prints `SportsCar created`.
Similarly, the invocation of the destructors takes place in the exact reverse order.

Question 4

Consider the following code segment. *[MCQ, Marks 2]*

```
#include <iostream>

using namespace std;

class Instrument {
public:
    void play() {
        cout << "Instrument::play()";
    }

    void play(int volume) {
        cout << "Instrument::play(int)";
    }
};

class Guitar : public Instrument {
public:
    void play() {
        cout << "Guitar::play()";
    }

    void play(int volume) {
        cout << "Guitar::play(int)";
    }
};

int main() {
    Guitar g;

    ______________________; // LINE-1

    return 0;
}
```

Choose the appropriate option to fill in the blank at `LINE-1` such that the output of the code becomes `Instrument::play()`.

a) `g.play()`

b) `Instrument::play()`

c) `g.Instrument::play()`

d) `Instrument::g.play()`

Answer: c)
Explanation:
We can access a base class function using the scope resolution operator even if it is hidden by a derived class function. Hence, c) is the correct option.

Question 5

Consider the following code segment. *[MCQ, Marks 2]*

```
#include <iostream>

using namespace std;

int value = 5;

class Parent {
protected:
    int value;

public:
    Parent() : value(15) {}

    ~Parent() {}
};

class Child : public Parent {
protected:
    int value;

public:
    Child() : value(25) {}

    ~Child() {}

    void display() {
        cout << ____________________ ; // LINE-1
    }
};

int main() {
    Child c;
    c.display();
    return 0;
}
```

Choose the appropriate option(s) to fill in the blank at `LINE-1` such that the output becomes `25 15 5`

a) `this->value << " " << Parent::value << " " << value`

b) `Child::value << " " << Parent::value << " " << value`

c) `value << " " << Parent::value << " " << ::value`

d) `Child::value << " " << Parent::value << " " << ::value`

Answer: c), d)
Explanation:
Since `value = 25` is in the scope of `class Child` which is also the local scope for the function

`display()`, `value` can be accessed by writing `Child::value` or just by `value`.
Since `value = 15` is in the scope of `class Parent`, it can be accessed by writing `Parent::value`.
Since `value = 5` is in the global scope, it can be accessed by writing `::value`. So option c) and d) both are correct.

Question 6

Consider the following code segment. *[MCQ, Marks 2]*

```
#include <iostream>

using namespace std;

int globalVar = 5;

class MyClass {
    int memberVar;

public:
    MyClass(int memberVar_ = 0) : memberVar(++memberVar_) { // LINE-1
        ++globalVar;
    }

    ~MyClass() {
        memberVar = 0;
        globalVar = 0;
    }

    void print() {
        cout << "memberVar = " << memberVar << ", globalVar = " << globalVar << endl;
    }
};

void test() {
    MyClass obj;
    obj.print();
}

int main() {
    MyClass obj;
    test();
    obj.print();
    return 0;
}
```

What will be the output?

a) `memberVar = 6, globalVar = 7`
   `memberVar = 1, globalVar = 8`

b) `memberVar = 6, globalVar = 8`
   `memberVar = 1, globalVar = 0`

c) `memberVar = 1, globalVar = 7`
   `memberVar = 2, globalVar = 8`

d) `memberVar = 1, globalVar = 7`
   `memberVar = 1, globalVar = 0`

Answer: d)
Explanation:
The statement `MyClass obj;` in `main()`, calls the constructor at `LINE-1` which makes `memberVar = 1` and `globalVar = 6`.
Then the statement `MyClass obj;` in `test()`, calls the constructor at `LINE-1` which makes `memberVar = 1` and `globalVar = 7`.
Then the statement `obj.print();` in `test()` prints `memberVar = 1`, `globalVar = 7`.
As the function `test()` returns, the destructor for the local object `obj` would be called, which makes `memberVar = 0` and `globalVar = 0`.
Finally, the statement `obj.print();` in `main()` prints `memberVar = 1`, `globalVar = 0`.

Question 7

Consider the following code segment. *[MCQ, Marks 2]*

```
#include <iostream>

using namespace std;

class Vehicle {
private:
    int reg_no;
    string type;

public:
    Vehicle(int reg_no_, string type_)
        : reg_no(reg_no_), type(type_) {}

    void showDetails() {
        cout << reg_no << ":" << type << ":";
    }
};

class Car : private Vehicle {
private:
    int owner_id;
    string owner_name;

public:
    Car(int owner_id_, string owner_name_, int reg_no_, string type_)
        : owner_id(owner_id_),
          owner_name(owner_name_),
          Vehicle(reg_no_, type_) {}

    void showDetails() {
        Vehicle::showDetails();  // LINE-1
        cout << owner_id << ":" << owner_name;
    }
};

int main() {
    Car* carObj = new Car(101, "Rahul", 12345, "Sedan");
    display(carObj);  // LINE-2
    delete carObj;
    return 0;
}
```

possible as, in vechicle showDetail is public and for next inheritance of car showDetail will be private

What will be the output/error?

a) `12345:Sedan:`

b) `12345:Sedan:101`

c) `compiler error at LINE-1:  Vehicle is an inaccessible base of Car`

d) `compiler error at LINE-2: ‘display’ was not declared in this scope`

Answer: d)
Explanation: Compilers will not convert a derived (`Car`) class object into a base (`Vehicle`) class object if the inheritance relationship is `private`. As the `display()` function takes a parameter of type class `Vehicle*` and we are passing a parameter of class `Car` object, it will give a compilation error as `inaccessible base`.

Question 8

Consider the following code segment. *[MCQ, Marks 2]*

```
#include <iostream>

using namespace std;

class Base {
public:
    Base() {}

    ~Base() {}

private:
    Base(const Base& obj) {}

    Base& operator=(const Base&) {}
};

class Derived : public Base {
    int data;

public:
    Derived() {}

    Derived(const int& data_) : data(data_) {}

    void print() {
        cout << data << " ";
    }
};

int main() {
    Derived d1(30);
    Derived d2(40);
    d1 = d2;
    d1.print();
    d2.print();
    return 0;
}
```

What will be the output/error?

a) `40 30`

b) `40 40`

c) `Compiler error: 'Base& operator=(const Base&)' is private`

d) `Compiler error: 'Base(const Base& obj)' is private`

Answer: c)
Explanation: Since `class Derived` inherits `class Base` where the copy constructor and

assignment operator function are both private, it prevents the free copy constructor and free assignment operator function from being added to the `Derived` class. As a result, when `d1 = d2;` invokes the assignment operator function, the call fails.

Question 9

Consider the following code segment. *[MCQ, Marks 2]*

```
#include <iostream>

using namespace std;

class X {
public:
    int x;
};

class Y : protected X {
public:
    int y;
};

class Z : public Y {
public:
    int z;

    Z(int x_, int y_, int z_) {
        x = x_;  // LINE-1
        y = y_;
        z = z_;
    }
};

int main() {
    Z zObj(10, 20, 30);
    cout << zObj.x << " ";  // LINE-2
    cout << zObj.y << " ";  // LINE-3
    cout << zObj.z;
    return 0;
}
```

What will be the output/error(s)?

a) `10 20 30`

b) `compiler error at LINE-1:  X::x is not accessible`

c) `compiler error at LINE-2:  X::x is not accessible`

d) `compiler error at LINE-3:  Y::y is not accessible`

Answer: c)
Explanation:
The inheritance relationship between `X` and `Y` is `protected`. Thus, `x` becomes `protected` in class `Y`. The inheritance relationship between `Y` and `Z` is `public`. Thus, `y` remains `public` in class `Y`, but `x` remains `protected` in class `Z`. As a result, `x` in `class Z` becomes invisible to the `main()` function. Hence, the correct answer is c).

Programming Questions

Question 1

Marks: 3

Consider the following program. Fill in the blanks as per the instructions given below:

- at `LINE-1` with the appropriate initialization list to initialize the data members,
- at `LINE-2` to call `displayContact()`, such that it will satisfy the given test cases.

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

class Contact {
    private:
        int phone_number;
        string email;
    public:
        Contact(int phone_number_, string email_) : phone_number(phone_number_),
                                                    email(email_){}
        void displayContact(){
            cout << "Phone: " << phone_number << endl;
            cout << "Email: " << email << endl;
        }
};

class Employee : private Contact {
    private:
        int emp_id;
        string emp_name;
    public:
        Employee(int emp_id_, string emp_name_, int phone_number_, string email_)
                        : ____________________{}    // LINE-1
        ___________________   // LINE-2
        void display(){
            cout << "ID: " << emp_id << endl;
            cout << "Name: " << emp_name << endl;
        }
};
int main(){
    int id, phone;
    string name, email;
    cin >> id >> name;
    cin >> phone >> email;
    Employee e(id, name, phone, email);
    e.display();
    e.displayContact();
    return 0;
}
```

Public 1

```
Input:
101 9876543210
Alice alice@example.com
Output:
ID: 101
Name: Alice
Phone: 9876543210
Email: alice@example.com
```

Public 2

```
Input:
102 1234567890
Bob bob@example.com
Output:
ID: 102
Name: Bob
Phone: 1234567890
Email: bob@example.com
```

Private

```
Input:
103 1112223333
Charlie charlie@example.com
Output:
ID: 103
Name: Charlie
Phone: 1112223333
Email: charlie@example.com
```

Answer:
`LINE-1: emp_id(emp_id_), emp_name(emp_name_), Contact(phone_number_, email_)`
`LINE-2: using Contact::displayContact;`
Explanation:
At `LINE-1`, the data members from `Employee` must be initialized as:
`emp_id(emp_id_), emp_name(emp_name_),` and the data members from `Contact` must be initialized as: `Contact(phone_number_, email_)` Although the function `displayContact()` is public in `Contact`, it becomes private in `Employee` due to private inheritance. So `displayContact()` becomes inaccessible using the `Employee` object. In order to call the `displayContact()` function using the `Employee` object, at `LINE-2` we must add:
`using Contact::displayContact;`

Question 2

Marks: 3

Consider the following program. Fill in the blanks as per the instructions given below:

- at `LINE-1` and `LINE-3` with the appropriate inheritance statement,
- at `LINE-2` and `LINE-4` with the appropriate initialization lists, such that it will satisfy the given test cases.

```
#include <iostream>

using namespace std;

class Appliance {
protected:
    int power;

public:
    Appliance(int p) : power(p) {}

    friend ostream& operator<<(ostream& os, const Appliance& a);
};

class WashingMachine : _________________{  // LINE-1
protected:
    int drum_size;

public:
    WashingMachine(int p, int ds) : __________ {}  // LINE-2

    friend ostream& operator<<(ostream& os, const WashingMachine& wm);
};

class Refrigerator : __________ {  // LINE-3
protected:
    int capacity;

public:
    Refrigerator(int p, int c) : ___________{}  // LINE-4

    friend ostream& operator<<(ostream& os, const Refrigerator& r);
};

ostream& operator<<(ostream& os, const Appliance& a) {
    os << "Power: " << a.power << "W" << endl;
    return os;
}

ostream& operator<<(ostream& os, const WashingMachine& wm) {
    os << "Power: " << wm.power << "W, Drum size: " << wm.drum_size << "L" << endl;
    return os;
}
```

```
ostream& operator<<(ostream& os, const Refrigerator& r) {
    os << "Power: " << r.power << "W, Capacity: " << r.capacity << "L" << endl;
    return os;
}

int main() {
    int a, b, c, d, e;
    cin >> a >> b >> c >> d >> e;
    Appliance appliance(a);
    WashingMachine wm(b, c);
    Refrigerator fridge(d, e);
    cout << appliance << wm << fridge;
    return 0;
}
```

Public 1

```
Input:
500 600 50 700 200
Output:
Power: 500W
Power: 600W, Drum size: 50L
Power: 700W, Capacity: 200L
```

Public 2

```
Input:
300 400 35 450 150
Output:
Power: 300W
Power: 400W, Drum size: 35L
Power: 450W, Capacity: 150L
```

Private

```
Input:
1000 850 60 950 250
Output:
Power: 1000W
Power: 850W, Drum size: 60L
Power: 950W, Capacity: 250L
```

Answer:

```
LINE-1:  public Appliance
LINE-2:  drum_size(ds), Appliance(p)
LINE-3:  public Appliance
LINE-4:  Appliance(p), capacity(c)
```

Explanation:
Since `WashingMachine` and `Refrigerator` both inherit `class Appliance`, at `LINE-1` and `LINE-3` the inheritance statement must be `public Appliance`.

At `LINE-2`, the initialization list must be:
`drum_size(ds), Appliance(p)`
and at `LINE-4`, the initialization list must be:
`Appliance(p), capacity(c)`

Question 3

Marks: 3

Consider the following program. Fill in the blanks as per the instructions given below:

- at `LINE-1, LINE-2` and `LINE-3` with initialization list,
- at `LINE-4, LINE-5` and `LINE-6` with function definitions, such that it will satisfy the given test cases.

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

class X {
    int x;
    public:
        X(int _x = 0);
        int getSum();
};

class Y : public X {
    int y;
    public:
        Y(int _x = 0, int _y = 0);
        int getSum();
};

class Z : public Y {
    int z;
    public:
        Z(int _x = 0, int _y = 0, int _z = 0);
        int getSum();
};

X::X(int _x) : _____________________________ {}    //LINE-1
Y::Y(int _x, int _y) : _____________________ {}    //LINE-2
Z::Z(int _x, int _y, int _z) : _____________ {}    //LINE-3
int X::getSum(){ ___________________________ }    //LINE-4
int Y::getSum(){ ___________________________ }    //LINE-5
int Z::getSum(){ ___________________________ }    //LINE-6

int main(){
    int a, b, c;
    cin >> a >> b >> c;
    X xObj(a);
    Y yObj(a, b);
    Z zObj(a, b, c);
    cout << xObj.getSum() << ", " << yObj.getSum() << ", " << zObj.getSum();
    return 0;
}
```

Public 1

```
Input: 10 20 30
```

```
Output: 10, 30, 60
```

Public 2

```
Input: 10 -10 10
Output: 10, 0, 10
```

Private

```
Input: 10 20 -10
Output: 10, 30, 20
```

Answer:

```
LINE-1:  x(_x)
LINE-2:  X(_x), y(_y)
LINE-3:  Y(_x, _y), z(_z)
LINE-4:  return x;
LINE-5:  return X::getSum() + y;
LINE-6:  return Y::getSum() + z;
```

Explanation:

The initialization lists at `LINE-1, LINE-2` and `LINE-3` are as follows:

```
LINE-1:  x(_x)
LINE-2:  X(_x), y(_y)
LINE-3:  Y(_x, _y), z(_z)
```

The definition of the `getSum()` functions at `LINE-4, LINE-5` and `LINE-6` are as follows:

```
LINE-4:  return x;
LINE-5:  return X::getSum() + y;
LINE-6:  return Y::getSum() + z;
```

Programming in Modern C++: Assignment Week 6

Total Marks : 25

Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur, Kharagpur – 721302
partha.p.das@gmail.com

August 20, 2024

Question 1

Consider the code segment given below. *[MCQ, Marks 2]*

```
#include<iostream>
using namespace std;
class Parent{
    public:
        void method1() { cout << "A" ; }
        virtual void method2() { cout << "C" ; }
};
class Child : public Parent{
    public:
        virtual void method1() { cout << "B" ; }
        void method2() { cout << "D" ; }
};
int main(){
    Parent *p = new Child();
    p->method1();
    p->method2();
    return 0;
}
```

Which line/s will give error?

a) `AC`

b) `AD`

c) `BC`

d) `BD`

Answer: b)
Explanation:
As `method1()` is a non-virtual function at the base class, for the `p->method1()` function call static binding is done. So, the function of pointer type will be called.
As `method2()` is a virtual function, for the `p->method2()` function call dynamic binding is done. So, the function of object type will be called.

Question 2

Consider the code segment given below. *[MCQ, Marks 2]*

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

class Animal {
    string type;
public:
    Animal(string t) : type(t) { }
    void displayType() {
        cout << type << " ";
    }
};

class Dog : public Animal {
    string breed;
public:
    Dog(string t, string b) : Animal(t), breed(b) { }
    void displayBreed() {
        cout << breed << " ";
    }
};

int main() {
    Dog d1("Mammal", "Labrador");
    Animal *a = &d1;
    a->displayType();  // LINE-1
    a->displayBreed(); // LINE-2
    return 0;
}
```

What will be the output/error?

a) `Mammal Labrador`

b) `Mammal 0`

c) `Compilation error at LINE-1: class Dog has no member named 'displayType'`

d) `Compilation error at LINE-2: class Animal has no member named 'displayBreed'`

Answer: d)
Explanation:
The object `d1` of class `Dog` is assigned to the pointer variable `a` of class `Animal`. The function `displayType()` is called using the pointer `a` which is valid. But `a` doesn't have a member function `displayBreed()`. Hence, `LINE-2` will give a compilation error.

Question 3

Consider the following code segment. *[MCQ, Marks 1]*

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

class X {
public:
    X() { cout << "X "; }
    ~X() { cout << "~X "; }
};

class Y : public X {
public:
    Y() { cout << "Y "; }
    virtual ~Y() { cout << "~Y "; }
};

class Z : public Y {
public:
    Z() { cout << "Z "; }
    ~Z() { cout << "~Z "; }
};

int main() {
    X *t1 = new Z;
    delete t1;
    return 0;
}
```

if x is virtual than all the destructor will invoke
O/P:- ~Z ~Y ~X

What will be the output?

a) `X Y Z ~Z ~Y ~X`

b) `X Y Z ~Z ~Y`

c) `X Y Z ~Y ~X`

d) `X Y Z ~X`

Answer: d)
Explanation:
When the object of class `Z` is created, it calls the constructor of class `Z`, which in turn calls the constructors of class `Y` and class `X` respectively. So, it will print `X Y Z`.
Whenever the object is deleted, it calls the destructor of class `X` first. The destructor of class `X` is not virtual, so it will not call the child class destructors. So, the final result will be `X Y Z ~X`.

Question 4

Consider the code segment given below. *[MCQ, Marks 1]*

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

class Base {
public:
    void display() {
        cout << "display()" << " ";
    }
};

class Derived : public Base {
public:
    ____________________// LINE-1
    void display(int) {
        cout << "display(int)" << " ";
    }
};

int main() {
    Derived d;
    d.display();
    return 0;
}
```

Fill in the blank at `LINE-1` such that the program runs successfully and prints `display()`

a) `Base::display();`

b) `using Base::display;`

c) `Base.display();`

d) `void display();`

Answer: b)
Explanation:
The function `display()` is called using the object of class `Derived`. But the function `display` is overloaded in `Derived`. Hence, the base class function `display()` becomes hidden in the derived class. To disable the method hiding in `Derived`, `LINE-1` will be filled as `using Base::display;`.

Question 5 *******

Consider the following code segment. *[MCQ, Marks 2]*

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

class Base {
public:
    Base() { cout << "B1 "; }
    virtual ~Base() { cout << "D1 "; }
};

class Derived : public Base {
public:
    Derived() { cout << "B2 "; }
    ~Derived() { cout << "D2 "; }
};

void func(Base b) {
    cout << "F ";
}

int main() {
    Derived d;
    func(d);
    return 0;
}
```

What will be the output?

a) `B1 B2 B1 F D1 D2 D1`

b) `B1 B2 F D1 D2 D1`

c) `B1 B2 F D2 D1`

d) `B1 B2 F D2`

Answer: b)
Explanation:
The object `d` invokes the constructors of class `Base` and `Derived` respectively, which prints `B1 B2`.
The argument of the function call invokes a copy constructor to pass the `Derived` object to the function. At the end of the function, the destructor of class `Base` gets invoked, which prints `D1` after printing `F`.
Similarly, the end of the `main()` function invokes the destructor of class `Derived` and class `Base` respectively, which prints `D2 D1`.

Question 6

Consider the following code segment. *[MSQ, Marks 2]*

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

class Shape {
public:
    ____________________ // LINE-1
};

class Circle : public Shape {
public:
    void draw() {
        cout << "Circle::draw" << endl;
    }
};

int main() {
    Shape *s = new Circle;
    s->draw();
    return 0;
}
```

Fill in the blank at `LINE-1` such that the program gives output as
`Circle::draw`

a) `void draw() = 0`

b) `virtual void draw()`

c) `virtual void draw() = 0`

d) `void draw()`

Answer: c)
Explanation:
Option a) and d) declare a non-virtual function, which requires a function definition (or function body). Thus, a) and d) are incorrect options.
Option b) is not a pure virtual function, which also needs the function definition.
Option c) is a pure virtual function, which does not require any definition. So it is the correct option.

Question 7

Consider the following code segment. *[MCQ, Marks 2]*

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

class Base {
    int b;
public:
    Base(int i) : b(i) {}
    virtual void display(Base *t) { cout << t->b << endl; }
};

class Derived : public Base {
    int d;
public:
    Derived(int i = 0, int j = 0) : Base(i), d(j) { }
    void display(Derived *t) { cout << t->d << endl; }
};

int main() {
    Base *ptr = new Derived(3, 4);
    ptr->display(new Derived); // Line-1
    return 0;
}
```

What will be the output?

a) `0`

b) `3`

c) `4`

d) `Garbage`

Answer: a)
Explanation:
The function call at LINE-1 invokes the base class function with a temporary object as a parameter. The temporary object of class Derived has data members initialized with default values (0). Hence, the program will print 0.

Question 8

Consider the following code segment. *[MCQ, Marks 2]*

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

class X {
public:
    virtual void display() { cout << "X::display()"; }
};

class Y : public X {
public:
    void display() override { cout << "Y::display()"; }
};

class Z : public Y {
public:
    void display() override { cout << "Z::display()"; }
};

int main() {
    Z obj;
    obj.Y::display();
    return 0;
}
```

What will be the output?

a) `X::display()`

b) `Y::display()`

c) `Z::display()`

d) `Compilation error: cannot call member function of class 'Y'`

Answer: b)
Explanation:
Since `obj` is of type `Z` and `class Z` inherits from `Y`, the statement `obj.Y::display()` calls the function `display()` of class `Y`, which prints `Y::display()`.

Question 9 *****

Consider the code segment given below. *[MCQ, Marks 2]*

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

class Base1 {
    public:
        virtual void show() { cout << "Base1::show"; }
        virtual void display() = 0;
        void print() { cout << "Base1::print"; }
};

class Base2 : public Base1 {
    public:
        void show() { cout << "Base2::show()"; }
        virtual void display() { cout << "Base2::display()"; }
        void print() { cout << "Base2::print"; }
};

int main() {
    Base2 b2;
    Base1 *p1 = &b2;
    Base1 &r1 = b2;

    p1->show();
    p1->display();
    p1->print();

    r1.show();
    r1.display();
    r1.print();

    return 0;
}
```

r1 will work for base1

What will be the output/error?

a) `Base2::show()Base2::display()Base1::printBase2::show()Base2::display()Base1::print`

b) `Base2::show()Base1::display()Base2::printBase2::show()Base2::display()Base1::print`

c) `Base2::show()Base2::display()Base1::printBase2::show()Base1::display()Base2::print`

d) `compile time error`

Answer: a)
Explanation:
The correct output is option **a)** because the code uses virtual functions for dynamic binding. The `show()` method is virtual in `Base1` and overridden in `Base2`, so `Base2::show()` is called for both `p1->show()` and `r1.show()` since `p1` and `r1` point to an object of `Base2`. The `display()` method, being pure virtual in `Base1` and overridden in `Base2`, also results in `Base2::display()` being called in both cases. However, the `print()` method is not virtual and

is overridden in `Base2`, but because it is non-virtual, the base class version `Base1::print()` is called in both cases (`p1->print()` and `r1.print()`), as the calls are resolved at compile time based on the type of the pointer or reference. This results in the final output of `Base2::show()Base2::display()Base1::printBase2::show()Base2::display()Base1::print`.

Programming Questions

Question 1

Marks: 3

Consider the following program. Fill in the blanks as per the instructions given below:

- at `LINE-1` with appropriate function declaration for `Area` function
- at `LINE-2` with appropriate function declaration for `Print` function such that it will satisfy the given test cases.

```
#include <iostream>
#define PI 3.14
using namespace std;

class Circle{
    protected:
        double radius;
    public:
        Circle(double r) : radius(r) {}
        ______________________          //LINE-1
        ______________________          //LINE-2
};

double Circle::Area() { return PI*radius*radius; }
void Circle::Print() { cout << Area() << " "; }

class Cylinder : public Circle{
    double height;
    public:
        Cylinder(double r, double h) : Circle(r), height(h) {}
        double Area() { return 2*PI*radius*radius*height; }
};

int main(){
    double r, h;
    cin >> r >> h;
    Circle c1(r);
    Cylinder c2(r,h);
    Circle *c[2] = {&c1, &c2};
    for(int i=0;i<2;i++)
        c[i]->Print();

    return 0;
}
```

Public 1

```
Input: 2 4
Output: 12.56 100.48
```

Public 2

```
Input: 5 1
Output: 78.5 157
```

Private

```
Input: 3 3
Output: 28.26 169.56
```

Answer:

```
LINE-1:  virtual double Area();
LINE-2:  void Print(); OR virtual void Print();
```

Explanation:

As the call to the `Area()` function needs to support dynamic binding, it must be declared as a virtual function.

The function `Print()` can be either a virtual or a non-virtual function.

Question 2

Marks: 3

Consider the following program. Fill in the blanks as per the instructions given below:

- at `LINE-1` with appropriate keyword,
- at `LINE-2` with appropriate statement so that `Mul()` can access private members of the class,
- at `LINE-3` with appropriate statement such that it will satisfy the given test cases.

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

class myClass1{
    int data1;
    public:
        myClass1(int d) : data1(d) {}
        _______ void print();                    //LINE-1
        ____________________________________;    //LINE-2
};

class myClass2 : public myClass1{
    int data2;
    public:
        myClass2(int d1, int d2) : myClass1(d1), data2(d2) {}
        void print(){
            _______________;                     //LINE-3
            cout << data2 << " ";
        }
};

void myClass1::print(){ cout << data1 << " "; }

void Mul(myClass1 &m1, myClass1 &m2){
    m1.data1 = m1.data1 * m2.data1;
}

int main(){
    int m, n;
    cin >> m >> n;
    myClass1 *t1 = new myClass2(m,n);
    myClass1 *t2 = new myClass1(m);
    Mul(*t1, *t2);
    t1->print();
    return 0;
}
```

Public 1

```
Input: 2 4
Output: 4 4
```

Public 2

```
Input: 3 5
Output: 9 5
```

Private

```
Input: 4 1
Output: 16 1
```

Answer:

```
LINE-1:  virtual
LINE-2:  friend void Mul(myClass1&, myClass1&)
LINE-3:  myClass1::print()
```

Explanation:

The `print()` function in the `myClass1` class should be declared as `virtual` such that the `myClass2` class function can be called using the `myClass1` class pointer type. The global function `Mul` can access private data members of the `myClass1` class only if it is a friend function of that class. So, `LINE-2` can be filled as `friend void Mul(myClass1&, myClass1&)`. The `myClass1` class `print` function can be called at `LINE-3` as `myClass1::print()`.

Question 3

Marks: 3

Consider the following program. Fill in the blanks as per the instructions given below:

- at `LINE-1` with appropriate statement to complete the `Base` class destructor declaration,
- at `LINE-2` with appropriate statement to complete the `Derived` class destructor declaration such that it will satisfy the given test cases.

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

class Base{
    int d;
    public:
        Base(int _d);
        _________________;        //LINE-1
        int get() { return d; }
};

class Derived : public Base{
    public:
        Derived(int _d);
        __________;               //LINE-2
};

Base::Base(int _d) : d(_d) { cout << d << " "; }
Base::~Base(){ cout << 2*d << " "; }
Derived::Derived(int _d) : Base(_d) { cout << 3*_d << " "; }
Derived::~Derived() { cout << Base::get() << " "; }

int main(){
    int m;
    cin >> m;
    Derived *t = new Derived(m);
    Base *t1 = t;
    delete t1;
    return 0;
}
```

Public 1

```
Input: 5
Output: 5 15 5 10
```

Public 2

```
Input: 2
Output: 2 6 2 4
```

Private

```
Input: 10
Output: 10 30 10 20
```

Answer:
`LINE-1:` `virtual ~Base()`
`LINE-2:` `~Derived() OR virtual ~Derived()`
Explanation:
From the test cases, it can be noticed that destructors from both the base and derived classes are being called while an object of the `Derived` class is deleted. So, we need to declare the base class destructor as `virtual`, which needs the `LINE-1` to be filled as `virtual ~Base();`. The `LINE-2` destructor declaration may or may not be `virtual`. So, it can be filled as `~Derived();` or `virtual ~Derived();`.

Programming in Modern C++: Assignment Week 7

Total Marks : 25

Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur, Kharagpur – 721302
partha.p.das@gmail.com

August 28, 2024

Question 1

Consider the following code segment. *[MSQ, Marks 2]*

```
#include <iostream>
using namespace std;
class employee {
    string name ;
    int salary;
    public:
        employee(int _sal, string _name) : name(_name), salary(_sal) {}
        void update(int s, string na) const{
            ( _________________________ )->salary = s; //LINE-1
            ( _________________________ )->name = na;  //LINE-2
        }
        void showInfo() const {
            cout << name << " : " << salary;
        }
};
int main(void) {
    const employee e(50000, "Ashim");
    e.update(30000, "Ram");
    e.showInfo();
    return 0;
}
```

Fill in the blank at `LINE-1` and `LINE-2` with the same statement such that the program will print `Ram :  30000`.

a) `const_cast <employee*> (this)`

b) `static_cast <employee*> (this)`

c) `dynamic_cast <employee*> (this)`

d) `(employee*)(this)`

Answer: a), d)
Explanation:

The statement `const employee e(50000, "Ashim");` defines `e` as a constant object. To modify its data-members, the constant-ness of the object need to be removed. This can be done either by `const_cast` (in option a) or casting constant this pointer to `employee*` type (in option d).

Question 2

Consider the following code segment. *[MCQ, Marks 1]*

```
#include<iostream>
using namespace std;
class A1{
    public:
        virtual void f() {}
        virtual void g() {}
};
class A2 : public A1{
    public:
        void g() {}
        void h() {}
        virtual void i();
};
class A3 : public A2{
    public:
        void f() {}
        virtual void h() {}
};
```

What will be virtual function table (VFT) for the `class A3`?

a) `A3::f(A3* const)`
   `A2::g(A2* const)`
   `A3::h(A3* const)`
   `A2::i(A2* const)`

b) `A1::f(A1* const)`
   `A2::g(A2* const)`
   `A3::h(A3* const)`
   `A2::i(A2* const)`

c) `A1::f(A1* const)`
   `A2::g(A2* const)`
   `A2::h(A2* const)`
   `A3::i(A3* const)`

d) `A1::f(A1* const)`
   `A2::g(A2* const)`
   `A3::h(A3* const)`
   `A3::i(A3* const)`

Answer: a)
Explanation:
All four functions are virtual in the `class A3`. So, there will be four entries in virtual function table.
Now, function `f()` is overridden in `class A3`. So, the entry for function `f()` in the virtual function table of `class A3` will be `A3::f(A3* const)`.
The function `g()` is virtual from class A1 and is overridden in `class A2`. So, the entry for function `g()` in VFT of `class A3` will be `A2::g(A2* const)`.
The function `h()` is declared as virtual in `class A3`. So, the entry for function `h()` in VFT of class C will be `A3::h(A3* const)`.

The function `i()` is declared as virtual in `class A2`. But not overridden in `A3`. So, the entry for function `i()` in VFT of `class A3` will be `A2::i(A2* const)`.

Question 3

Consider the following code segment. *[MSQ, Marks 2]*

```
#include <iostream>
using namespace std;
int main() {
    char ch = 'C';
    double db = 3.14;
    char *cp = &ch;
    double *pd;
    ch = static_cast<char>(db);      // LINE-1
    db = static_cast<double>(ch);    // LINE-2
    pd = static_cast<double*>(cp); // LINE-3
    ch = static_cast<char>(&ch);     // LINE-4
    return 0;
}
```

Which line/s will give compilation error?

a) `LINE-1`

b) `LINE-2`

c) `LINE-3`

d) `LINE-4`

Answer: c), d)
Explanation:
`static_cast` cannot cast between two different pointer types. In `LINE-3`, `double*` is assigned to `char*`. Hence it is error.
Using static cast, it is not possible to change a pointer type to a value type. In `LINE-4`, `char*` is assigned to `char` which is not possible using static cast.

Question 4

Consider the following code segment. *[MCQ, Marks 1]*

```
class Test1 { };
class Test2 { };
Test1* tst1 = new Test1;
Test2* tst2 = new Test2;
```

Which of the following type-casting is permissible?

a) `tst2 = static_cast<Test2*>(tst1);`

b) `tst2 = dynamic_cast<Test2*>(tst1);`

c) `tst2 = reinterpret_cast<Test2*>(tst1);`

d) `tst2 = const_cast<Test2*>(tst1);`

Answer: c)
Explanation:
On each option, there is an attempt to cast from `Test1*` to `Test2*`, and these two classes are unrelated. As we know, only `reinterpret_cast` can be used to convert a pointer to an object of one type to a pointer to another object of an unrelated type. Hence only option c) is correct.

Question 5

Consider the following code segment. *[MCQ, Marks 2]*

```
#include <iostream>
#include <typeinfo>
using namespace std;
class C1 { public: virtual ~C1(){}};
class C2: public C1{};
int main() {
    C1 b;
    C2 d;
    C2 *dp = &d;
    C1 *bp = dp;
    C2 *dpp = (C2*)dp;
    cout << (typeid(bp).name() == typeid(dpp).name());
    cout << (typeid(*bp).name() == typeid(*dpp).name());
    cout << (typeid(dp).name() == typeid(dpp).name());
    cout << (typeid(*dp).name() == typeid(*dpp).name());
    return 0;
}
```

What will be the output?

a) `0101`

b) `0111`

c) `0110`

d) `0010`

Answer: b)
Explanation:
Type of `bp` is `C1*` and type of `dpp` is `C2*`. Thus, output is 0.
`*bp` and `*dpp` point to the same object d, and it is a dynamic binding situation. Thus, both are of type C2 and output is 1.
Type of `dp and dpp` is `C2*`. Thus, output is 1.
`*dp and *dpp` point to the same object d, and it is a dynamic binding situation. Thus, both are of type C2 and output is 1.

Question 6

Consider the following code segment. *[MCQ, Marks 2]*

```
#include <iostream>
using namespace std;
class A1{ public: virtual ~A1(){} };
class A2 : public A1{};
class A3 : public A1{};
int main(){
    A1 objA;
    A2 objB;
    A1* pA = dynamic_cast<A1*>(&objB); //LINE-1
    pA == NULL ? cout << "0" : cout << "1";
    A2* pB = dynamic_cast<A2*>(pA); //LINE-2
    pB == NULL ? cout << "0" : cout << "1";
    A3* pC = dynamic_cast<A3*>(new A1); //LINE-3
    pC == NULL ? cout << "0" : cout << "1";
    pC = dynamic_cast<A3*>(&objB); //LINE-4
    pC == NULL ? cout << "0" : cout << "1";
    return 0;
}
```

What will be the output?

a) `0101`

b) `1010`

c) `1100`

d) `1011`

Answer: c)
Explanation:
The type-casting at `LINE-1` is valid as it is an upper-casting.
At `LINE-2`, though it is a down-casting, it is allowed as the pointer `pB` points to the same type of object (which of type A2).
At `LINE-3`, the down-casting is invalid as the pointer pC points to parent type of object (which is of type A1). Hence prints 0.
At `LINE-4`, the casting is also invalid as the pointer pC points to an object (which is of type A2) that is neither of its base type or derived type and hence prints 0.

Question 7

Consider the following code segment. *[MSQ, Marks 2]*

```
#include <iostream>
using namespace std;
int main() {
    const double g = 9.81;
    const double *pg = &g;
    double *pt = ______________________(pg); //LINE-1
    *pt = 10;
    cout << *pt;
    return 0;
}
```

Fill in the blank at `LINE-1` so that the program will print `10`.

a) `const_cast<double*>`

b) `static_cast<double*>`

c) `dynamic_cast<double*>`

d) `(const double*)`

Answer: a)
Explanation:
At `LINE-1`, pg of type `const double*` needs to be casted to `double*`. It can be accomplished by `const_cast<double*>(pg)`.
This question is intentionally made as MSQ

Question 8

Consider the following code segment. *[MSQ, Marks 2]*

```
#include<iostream>
using namespace std;
class A{
    string a = "C++";
};
class B{
    public:
        string b = "Java";
};
int main(){
    A u;
    B *v = ____________________________(&u);
    cout << v->b;
    return 0;
}
```

Fill in the blank at `LINE-1` so that the program will print "C++".

a) `reinterpret_cast<B*>`

b) `static_cast<B*>`

c) `dynamic_cast<B*>`

d) `(B*)`

Answer: a), d)
Explanation:
We need to cast object of `A` to `B` type which are unrelated. This can be done using C style casting or reinterpret cast.

Question 9

Consider the code segment given below. *[MCQ, Marks 2]*

```
class C1{ public: void f(){} };
class C2 : public C1 { public: virtual void f(){}};
class C3 : public C2{ public: void g(){} };
class C4 : public C1{ public: virtual void g(){} };
```

How many virtual function table (VFT) will be created?

a) `1`

b) `2`

c) `3`

d) `4`

Answer: c)
Explanation:
The presence of a virtual function (either explicitly declared or inherited from a base class) makes the class polymorphic. For such classes, we need a class-specific virtual function table (VFT). All three classes except `A1`, thus, will set up virtual function tables.

Programming Questions

Question 1

Complete the program with the following instructions.

- Fill in the blank at `LINE-1` to complete operator overloading for the assignment operator.
- Fill in the blanks at `LINE-2` and `LINE-3` to complete the type casting statements.

The program must satisfy the given test cases. *Marks: 3*

```
#include<iostream>
using namespace std;
class Class1{
    int a = 10;
    public:
        void show(){
            cout << a << " ";
        }
};
class Class2{
    int b = 20;
    public:
        void show(){
            cout << b;
        }
        ______________________{ //LINE-1
            b = b + x;
        }
};
void fun(const Class1 &t, int x){
    Class1 &u = ____________________(t); //LINE-2
    u.show();
    Class2 &v = ___________________________(u); //LINE-3
    v = x;
    v.show();
}
int main(){
    Class1 t1;
    int x;
    cin >> x;
    fun(t1, x);
    return 0;
}
```

Public 1

```
Input: 4
Output: 10 14
```

Public 2

```
Input: 9
Output: 10 19
```

Private 1

```
Input: 15
Output: 10 25
```

Answer:

```
LINE-1:  operator=(int x)
LINE-2:  const_cast<Class1&>
LINE-3:  reinterpret_cast<Class2&>
```

Explanation:

As per the function `fun()`, we need to overload the operator equal to for the class `Class2` at `LINE-1` so that the assignment `v = x` will be valid. It can be done as `operator=(int x)`.

To call a non-constant function `show()` using a const object reference u, we need to cast the reference to a non-const reference. So, `LINE-2` will be filled as `const_cast<Class1&>`.

Casting between two unrelated classes at `LINE-3` can be done as `reinterpret_cast<Class2&>`.

Question 2

Consider the following program with the following instructions.

- Fill in the blank at `LINE-1` to complete constructor definition.
- Fill in the blank at `LINE-2` to complete assignment operator overload function signature.
- Fill in the blank at `LINE-3` to complete integer cast operator overload function signature.

The program must satisfy the sample input and output. *Marks: 3*

```
#include<iostream>
using namespace std;
class TestClass{
    int *arr;
    int n;
    public:
        TestClass(int k) : ______________________{} //LINE-1
        ________________{ //LINE-2
            return arr[--n];
        }
        ___________________(int &k){ //LINE-3
            int t;
            for(int j = 0; j < k; j++){
                cin >> t;
                this->arr[j] = t;
            }
            return *this;
        }
};
int main(){
    int k;
    cin >> k;
    TestClass s(k);
    s = k;
    for(int i = 0; i < k; i++)
        cout << static_cast<int>(s) << " ";
    return 0;
}
```

Public 1

```
Input: 3
1 2 3
Output: 3 2 1
```

Public 2

```
Input: 4
5 6 4 7
Output: 7 4 6 5
```

Private

```
Input: 5
1 2 3 4 5
Output: 5 4 3 2 1
```

Answer:
`LINE-1:` `n(k), arr(new int(n))`
`LINE-2:` `operator int()`
`LINE-3:` `TestClass operator=`
Explanation:
The initialization of the data-members at `LINE-1` can be done as:
`n(k), arr(new int(k))`
At `LINE-2`, we overload type-casting operator for the statement `static_cast<int>(s)` as:
`operator int()`
At `LINE-3`, we overload `operator=` for the statement `s = k;` as:
`TestClass operator=(int& k)`

Question 3

Consider the following program. Fill in the blanks as per the instructions given below:

- at `LINE-1` to complete the constructor definition,
- at `LINE-2` to complete the function header to overload operator for type-casting to `char`,
- at `LINE-3` to complete the function header to overload operator for type-casting to `int`,

such that it will satisfy the given test cases. *Marks: 3*

```
#include<iostream>
#include<cctype>
using namespace std;
class Character{
    char c;
    public:
        _____________ : c(tolower(_c)){} //LINE-1
        _____________{ return c; } //LINE-2
        _____________{ return c - 'a' + 1; } //LINE-3
};
int main(){
    char c;
    cin >> c;
    Character cb = c;
    cout << (char)cb << ": position is " << int(cb);
    return 0;
}
```

Public 1

```
Input: A
Output: a: position is 1
```

Public 2

```
Input: c
Output: c: position is 3
```

Private

```
Input: Z
Output: z: position is 26
```

Answer:
`LINE-1:` `Character(char _c)`
`LINE-2:` `operator char()`
`LINE-3:` `operator int()`
Explanation:
The statement `Char cb = c;` requires casting from char to class `Character`, which is implemented by the constructor as:
`Character(char _c)`
The statement `(char)cb` requires a casting operator from class `Character` to char which may be done by the function

`operator char()`
The statement `int(cb)` requires a casting operator from class `Character` to int which may be done by function
`operator int()`

Programming in Modern C++: Assignment Week 8

Total Marks : 27

Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur, Kharagpur – 721302
partha.p.das@gmail.com

September 5, 2024

Question 1

Consider the program given below. *[MCQ, Marks 2]*

```
#include<iostream>

void validate(int i){
    try{
        if(i < 0)
            throw "negetive";
        else if(i > 0)
            throw i * 1.00;
        else
            throw 0;
    }
    catch(float ){ std::cout << "float "; }
    catch(int ){ std::cout << "int "; }
    catch(const char* ){ std::cout << "char *"; }
}
int main(){
    try{
        validate(0);
        validate(5);
        validate(-5);
    }
    catch(...) { std::cout << "ALL"; } //LINE-4
}
```

throw double not float

What will be the output?

a) `int float char *`

b) `int float`

c) `int ALL`

d) `int ALL char *`

Answer: c)
Explanation:
For call `validate(0);` it executes statement `throw 0;`, which throws `int` type exception. `int` type exception will be handled within the function `validate()` and it prints `int` .
For call `validate(5);` it executes statement `throw i * 1.00;`, which throws `double` type exception. `double` type exception will be handled in the `main` function (because `validate()` cannot handle it) and it prints `ALL` . Due to the exception in `main()` the control comes out of the `try` block.

Question 2

Consider the program given below. *[MCQ, Marks 1]*

```
#include<iostream>

void func(){
    try{
        throw 9.81;
    }
    catch(int& e){ throw e; }
}

int main(){
    try{
        func();
    }
    catch(int& e){ std::cout << "int"; }
    catch(...){ std::cout << "all"; }            //LINE-1
    catch(double& e){ std::cout << "double"; }
    return 0;
}
```

What will be the output/error?

a) `all`

b) `int`

c) `double`

d) `Compiler error LINE-1: '...' handler must be the last handler for its try block`

Answer: d)
Explanation:
`catch(...) { ... }` must be the last handler for its try block. Thus, it generates an error at `LINE-1`.

Question 3

Consider the following program. *[MCQ, Marks 2]*

```
#include<iostream>

class Printer {};
class Nozzle : public Printer{};
class Cartridge : public Printer{};

void print(int i){
    try{
        i < 0 ? throw Nozzle() : throw new Cartridge();
    }
    catch(Printer) { std::cout << "PrinterException "; }
    catch(Nozzle) { std::cout << "NozzleException "; }
    catch(Cartridge) { std::cout << "CartridgeException "; }
};


int main(){
    try{
        for(int i = -1; i < 2; i++)
            print(i);
    }
    catch(...) { std::cout << "Some other exception "; }
    return 0;
}
```

Cartridge* which is not present in catch

catch of Printer will called because of base class

What will be the output?

a) `NozzleException CartridgeException CartridgeException`

b) `PrinterException Some other exception`

c) `PrinterException PrinterException`

d) `NozzleException Some other exception`

Answer: b)
Explanation:
For `print(-1)`, the thrown exception type is `Nozzle*`. Since within `print()` function, the first catch block handles the exception of type `Printer` that is base class of `Nozzle`, the first catch block will handle it and the output will be `PrinterException`.
For `print(0)`, the thrown exception type is `Cartridge*`. Since it is not handled within `print()` function, it will be forwarded to `main()` function. In `main()`, it will be handled with printing `Some other exception`, and the control goes out of `for` loop.

Question 4

Consider the following program. *[MCQ, Marks 2]*

```
#include<iostream>
namespace DBErrors{
class SQLException {};
class KeyException : public SQLException {};
class PrimaryKeyException : public KeyException{};
class ForeignKeyException : public KeyException{};
class DBCon{
public:
    static void print_err(int eno = 0){
        try{
            if(eno == 0)
                throw PrimaryKeyException();
            else if(eno < 0)
                throw ForeignKeyException();
            else if (eno > 0 && eno < 5)
                throw KeyException();
            else
                throw SQLException();
        }
        catch(KeyException&) { std::cout << "DBErrors::KeyException"; }
    }
};
}
int main(){
    try{
        ____________________________________; //LINE-1
    }
    catch(DBErrors::PrimaryKeyException&) {
        std::cout << "DBErrors::PrimaryKeyException&";
    }
    catch(DBErrors::ForeignKeyException&) {
        std::cout << "DBErrors::ForeignKeyException";
    }
    catch(DBErrors::SQLException) {
        std::cout << "DBErrors::SQLException";
    }
    return 0;
}
```

only one previous base class have to check

base class of this is KeyException

base class of this is KeyException

Identify the option/s to fill in the blank at `LINE-1` such that output **IS NOT** `DBErrors::KeyException`.

a) `DBErrors::DBCon::print_err(-10)`

b) `DBErrors::DBCon::print_err(0)`

c) `DBErrors::DBCon::print_err(7)`

d) `DBErrors::DBCon::print_err(20)`

Answer: c), d)
Explanation:
For option c) and d), it throws exception `SQLException()`, which is not handled by the catch block within the function `print_err()`. It would be forwarded to `main` function, where it will handed by printing `DBErrors::SQLException`.
For all other options, the exception thrown would be handled within the function `print_err()` since `KeyException` is the base class of `PrimaryKeyException` and `ForeignKeyException`, and it will print `DBErrors::KeyException`.

Question 5

Consider the following code segment. *[MSQ, Marks 2]*

```
#include<iostream>
template<typename S>
int compare(S n1, S n2){
    return n1 * n2;
}
int main(){
    std::cout << compare(31.46, 34); //LINE-1
    return 0;
}
```

Which of the following call/s to `compare` at `LINE-1` will result in compiler error?

a) `compare('i', 'k')`

b) `compare(31.46, 34.0)`

c) `compare(31.46, 34)`

d) `compare('A', 34)`

Answer: c), d)
Explanation:
Since the types of the actual arguments in the calls in option c) and d) are different, the deduction of type `S` is ambiguous. Therefore, it generates a compiler error – "no matching function for call to compare".

Question 6

Consider the following code segment. *[MCQ, Marks 2]*

```
#include<iostream>
_____________________________________________ //LINE-1
class TempTest{
    T1 a_;
    T2 b_;
public:
    TempTest(T1 a, T2 b){
        a_ = a;
        b_ = b;
    }
    void test(){
        std::cout << "a = " << a_ << ", b = " << b_ << std::endl;
    }
};
int main(){
    TempTest<int> t1(97, 65);
    TempTest<> t2(97, 65);
    t1.test();
    t2.test();
    return 0;
}
```

Fill in the blank at `LINE-1` such that the output of the program is:

```
a = 97, b = 65
a = a, b = 65
```

a) `template<typename T1, typename T2>`

b) `template<typename T1 = char, typename T2 = int>`

c) `template<typename T1=int, typename T2>`

d) `template<typename T1=int, typename T2=char>`

Answer: b)
Explanation:
From the output, it can be concluded that the default type of `T1` is char and `T2` is `char`. Thus, option b) is correct.

Question 7

Consider the following program. *[MCQ, Marks 1]*

```
#include <iostream>
template <typename T, int N> //LINE-1
class NPrint{
    T _v;
public:
    NPrint(T v) : _v(v){}
    void print(){
        for(int i = 0; i < N; i++)
            std::cout << _v;
    }
};
int main(){
    int n = 3;
    NPrint<char, n> np('C'); //LINE-2
    np.print();
    return 0;
}
```

const int n = 3; for correct output

What will be the output / error?

a) `333`

b) `CCC`

c) `Compiler error at LINE-1: non-type argument is not allowed`

d) `Compiler error at LINE-2: the value of 'n' is not usable in a constant expression`

Answer: d)
Explanation:
In C++ template, non-type argument n in template must be a constant.

Question 8

Consider the following code segment. *[MCQ, Marks 2]*

```
#include<iostream>
class complex_num{
    int re_, im_;
public:
    complex_num(int re = 0, int im = 0) : re_(re), im_(im){}
    int getRe(){ return re_; }
    int getIm(){ return im_; }
    void setRe(int re){ re_ = re; }
    void setIm(int im){ im_ = im; }
    friend std::ostream& operator<<(std::ostream& os, const complex_num& c);
};
std::ostream& operator<<(std::ostream& os, const complex_num& c){
    os << c.re_ << " + i" << c.im_ << std::endl;
    return os;
}
template<class T> T add(T x, T y){
    return x + y;
}
------------------------------------------------------------ { //LINE-1
    complex_num t;
    t.setRe(x.getRe() - y.getRe());
    t.setIm(x.getIm() + y.getIm());
    return t;
}
int main(){
    complex_num c1(10, 10);
    complex_num c2(5, 15);
    std::cout << add<double>(5.5, 1.4) << ", ";
    std::cout << add<int>(5, 4) << ", ";
    std::cout << add<complex_num>(c1, c2);
    return 0;
}
```

Identify the appropriate option to fill in the blank at `LINE-1` such that the program gives output as
`6.9, 9, 5 + i25`

a) `complex_num add(complex_num x, complex_num y)`

b) `template<complex_num> complex_num add(complex_num x, complex_num y)`

c) `template<> complex_num add<complex_num>(complex_num x, complex_num y)`

d) `template<T> add<complex_num>(complex_num x, complex_num y)`

Answer: c)
Explanation:
The appropriate statement for template specification for `complex_num` at `LINE-1` is:
`template<> complex_num add<complex_num>(complex_num x, complex_num y))`

Question 9

Consider the following code segment. *[MCQ, Marks 2]*

```
#include <iostream>

class computation {
   public:
      int addition(int a, int b) {
         return a + b;
      }
      int subtraction(int a, int b) {
         return a - b;
      }
};

_____________________________________________________ {    //LINE-1
   return (obj->*fp)(x, y);
}

int main() {
   computation cm;
   std::cout << caller(10, 20, &cm, &computation::addition) << " ";
   std::cout << caller(10, 20, &cm, &computation::subtraction) << std::endl;
   return 0;
}
```

Choose the appropriate option to fill in the blank at `LINE-1` so that the output becomes `30 -10`

a) `int caller(int& x, int& y, computation* obj, int(computation::fp)(int, int))`

b) `int caller(int x, int y, computation* obj, int(computation::*fp)(int, int))`

c) `int caller(int& x, int& y, computation* obj, int(*computation::fp)(int, int))`

d) `int caller(int x, int y, computation* obj, int*(computation::fp)(int, int))`

Answer: b)
Explanation:
The appropriate way to declare a function pointer to point to `computation::addition` and `computation::subtraction` is as follows:
`int(computation::*fp)(int, int))`
Therefore, the correct option is b).

Programming Questions

Question 1

Consider the program below.

- Fill in the blank at `LINE-1` with appropriate template definition for function `average`.
- Fill in the blank at `LINE-2` with appropriate header for function `average`.
- Fill in the blank at `LINE-3` with appropriate declaration of `size`.

The program must satisfy the given test cases. *Marks: 3*

```
#include <iostream>
_____________________________________ //LINE-1
_____________________________ { //LINE-2
    T1 total = 0 ;
    for(int i = 0; i < N; i++)
        total += arr[i];
    r = (double)total / N;
}
int main(){
    ____________ size = 4; //LINE-3
    int iA[size];
    for(int i = 0; i < size; i++)
        std::cin >> iA[i];
    double avg = 0.0;
    average<int, double, size>(iA, avg);
    std::cout << avg;
    return 0;
}
```

Public 1

```
Input: 10 20 30 40
Output: 25
```

Public 2

```
Input: -2 4 5 -1
Output: 1.5
```

Private

```
Input: 9 10 -5 7
Output: 5.25
```

Answer:

```
LINE-1:  template<typename T1, typename T2, int N>
```

Or

```
LINE-1:  template<class T1, class T2, int N>
LINE-2:  void average(T1 arr[], T2& avg)
LINE-3:  const int
```

Explanation:

Since the call to the function `average` is made as:
`average<int, double, 5>(iA, avg);` , the first two arguments of template are type arguments, whereas the third one is non-type, which can be filled at `LINE-1` as:
`template<typename T1, typename T2, int N>`
Or
`template<class T1, class T2, int N>`
At `LINE-2`, the function header can be
`void average(T1 arr[], T2& avg)`
At `LINE-3`, the constant `size` can be declared as:
`LINE-3:  const int`

Question 2

Consider the following program that computes the mean of the squares of a number of integers. Fill in the blanks as per the instructions given below to complete the definition of functor `average` with local state:

- Fill in the blank at `LINE-1` with constructor header.
- At `LINE-2` overload the function call operator.

The program must satisfy the sample input and output. *Marks: 3*

```
#include<iostream>
#include<algorithm>
#include<vector>
struct average {
    int cnt_; //count of element
    int ss_; //sum of square of emements
    ______________________________ : cnt_(cnt), ss_(ss) {} //LINE-1
    ____________________________________________ //LINE-2
};
int main(){
    std::vector<int> v;
    int n, a;
    std::cin >> n;
    for(int i = 0; i < n; i++){
        std::cin >> a;
        v.push_back(a);
    }
    average mi = for_each(v.begin(), v.end(), average());
    std::cout << "mean = " << (double)mi.ss_/mi.cnt_;
    return 0;
}
```

Public 1

```
Input: 4 1 2 3 4
Output: mean = 7.5
```

Public 2

```
Input: 5 10 20 30 40 50
Output: mean = 1100
```

Private

```
Input: 4 10 9 8 7
Output: mean = 73.5
```

Answer:

```
LINE-1:  average(int cnt = 0, int ss = 0)
LINE-2:  void operator() (int x) { ++cnt_ ; ss_ += x * x; }
```

Explanation:
The constructor must initialize `ss = 0` and `cnt = 0`. Therefore, the constructor header must

be defined as:

```
max(int cnt = 0, int ss = 0)
```

The function call operator must be overloaded to find the count of the elements of the array, and to get the sum of the squares of the elements. Thus, it must be:

```
void operator() (int x) { ++cnt_; ss_ += x * x; }
```

Question 3

Consider the following program, which define a type `plist` that stores N elements. Fill in the blanks as per the instructions given below:

- Fill in the blank at `LINE-1` with appropriate template definition,
- At `LINE-2` implement `void insert(int i, T val)` function, which inserts a value at specified index; however, if the input index is $\geq N$, it throws `testArray` exception,
- At `LINE-3` implement void `T peek(int i)` function, which returns value of a specified index. However, if the element at the specified position is negative then it thorws `InvalidElement` exception.

The program must satisfy the sample input and output. *Marks: 3*

```
#include<iostream>
class testArray{
public:
    virtual void what(){ std::cout << "index out of array"; }
};
class InvalidElement{
public:
    virtual void what(){ std::cout << "invalid"; }
};
____________________________________ //LINE-1
class plist{
    T arr_[N];
public:
    plist(){
        for(int i = 0; i < N; i++)
            arr_[i] = -1;
    }
    //LINE-2: impelement insert() function
    //LINE-3: impelement peek() function
};
int main(){
    int n;
    char c;
    plist<char, 4> li;
    try{
        for(int i = 0; i < 4; i++){
            std::cin >> n;
            std::cin >> c;
            li.insert(n, c);
        }
    }catch(testArray& e){
        e.what();
        std::cout << std::endl;
    }
    for(int i = 0; i < 4; i++){
        try{
            std::cout << li.peek(i) << ", ";
        }catch(InvalidElement& e){
```

```
            e.what();
            std::cout << ", ";
        }
    }
    return 0;
}
```

Public 1

```
Input: 1 a 2 b 3 c 0 x
Output: x, a, b, c,
```

Public 2

```
Input: 2 a 1 x 3 z 2 y
Output: invalid, x, y, z,
```

Public 3

```
Input: 0 a 2 b 3 x 4 z
Output:
index out of array
a, invalid, b, x,
```

Public 4

```
Input: 0 a 6 x
Output:
index out of array
a, invalid, invalid, invalid,
```

Private

```
Input: 0 x 2 y 3 z 2 a
Output: x, invalid, a, z,
```

Answer:

```
LINE-1:  template<typename T, int N>
```

or

```
LINE-1:  template<class T, int N>
LINE-2:
```

```
void insert(int i, T val){
    if(i < N)
        arr_[i] = val;
    else
        throw testArray();
}
```

```
LINE-3:
```

```
T peek(int i){
    if(arr_[i] < 0)
        throw InvalidElement();
    return arr_[i];
}
```

Explanation:
At `LINE-1`, the template definition must be:
`template<typename T, int N>`
or
`LINE-1: template<class T, int N>`
At `LINE-2`, define the `void insert(int i, T val)` to add element `val` at index `i` as follows:

```
void insert(int i, T val){
    if(i < N)
        arr_[i] = val;
    else
        throw testArray();
}
```

At `LINE-3`, define the `T peek(int i)` to return the element at index `i` as follows:

```
T peek(int i){
    if(arr_[i] < 0)
        throw InvalidElement();
    return arr_[i];
}
```

Programming in Modern C++: Assignment Week 9

Total Marks : 27

Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur, Kharagpur – 721302
partha.p.das@gmail.com

September 12, 2024

Question 1

Consider the program given below. *[MCQ, Marks 2]*

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

int main () {
    int i = 90;
    cout << setbase(8)<< i << " ";
    cout << setbase(10)<< i << " ";
    cout << setbase(16)<< i << " ";
    cout << static_cast<char>(i) << endl;
    return 0;
}
```

What will be the output?

a) `90 90 90 90`

b) `90 90 90 Z`

c) `132 90 5a 90`

d) `132 90 5a Z`

Answer: d)
Explanation:
The function `setbase(b)` sets the base of an integer as `b`. Therefore, the statement :
`cout << setbase(8)<< i << " ";`
prints the value of `i` as octal (base 8) value.
The statement:
`cout << setbase(10)<< i << " ";`
prints the value of `i` as decimal (base 10) value.
The statement:
`cout << setbase(16)<< i << " ";`
prints the value of `i` as octal (base 16) value.
Finally, the statement `static_cast<char>(i)` type-casts `i` to integer.

Question 2

Consider the program given below. *[MCQ, Marks 2]*

```
#include<cstdio>
using namespace std;
int main(){
    FILE *inp, *outp;
    int c;
    if((inp = fopen("in.txt", "r")) == NULL)
        return -1;
    if((outp = fopen("out.txt", "w")) == NULL)
        return -2;
    while((c = fgetc(inp)) != EOF && c != '\n')
        fputc(c, outp);
    fclose(inp);
    fclose(outp);
    return 0;
}
```

Choose the statement that is true about the output of the program.

a) It copies the entire content file `in.txt` to file `out.txt`.

b) It copies the content file `in.txt` to file `out.txt` without the newlines (i.e. the entire content of `in.txt` would be copied to a single line of `put.txt`.

c) It copies the first line of file `in.txt` to file `out.txt`.

d) It copies the content file `in.txt` to file `out.txt` if file `in.txt` has only one line; otherwise, it generates an error in runtime.

Answer: c)
Explanation:
The `while` loop in the program gets terminated either when `fgetc(inp)` return `EOF` or `c == '\n'` . Therefore, if the file `in.txt` has only one line, the `EOF` will be encountered at the end of the line. In case of multiple lines, it reads the first line, then encounters a `'\n'`, and terminates.

Question 3

Consider the following code segment. *[MCQ, Marks 1]*

```
#include <iostream>
#include <fstream>

int main () {
    std::ifstream inf("input.txt");
    std::string line;
    if (__________________) { //LINE-1
        std::cout << "file is not opened";
    }
    else{
        while (getline(inf, line))
            std::cout << line << std::endl;
        inf.close();
    }
    return 0;
}
```

Choose the appropriate option to fill in the blank at `LINE-1` such that it checks if the file `input.txt` is opened or not.

a) `inf.is_open()`

b) `!inf.is_open()`

c) `!inf.open()`

d) `fopen(inf) == NULL`

Answer: b)
Explanation:
Since the program attempts to read from the file, whether the file exists or not can be verified by `is_open()` function. Thus, the correct option is b).

Question 4

Consider the following code segment. *[MSQ, Marks 2]*

```
#include<iostream>

template<class Itr, class T>
int findmax(Itr first, Itr last, T& mval) {
    int maxpos = 0, i = 0;
    mval = *first++;
    while (first != last) {
        if(*first > mval){
            mval = *first;
            maxpos = i + 1;
        }
        ++first;
        ++i;
    }
    return maxpos;
}

int main(){
    int iArr[] = { 1, 4, 8, 1, 3, 8, 7};
    double mVal = 0.0;
    ____________________________________________________; //LINE-1
    std::cout << pos << ", " << mVal;
    return 0;
}
```

Choose the appropriate options to fill in the blank at `LINE-1` such that the program finds out the maximum element of the array `iArr` and the output is `2, 8`.

a) `findmax(iArr, iArr + sizeof(iArr) / sizeof(*iArr), mVal)`

b) `int pos = findmax(iArr, &iArr[sizeof(iArr) / sizeof(*iArr)], mVal)`

c) `int mVal = findmax(iArr, iArr + sizeof(iArr) / sizeof(*iArr), mVal)`

d) `int pos = findmax(iArr, iArr + sizeof(iArr) / sizeof(*iArr), mVal)`

Answer: b), d)
Explanation:
The formal parameters of the function `findmax` are as follows:

- `first` – the base address of the array, which is iArr
- `last` – the end address of the array, which is `iArr + sizeof(iArr) / sizeof(*iArr)` or `&iArr[sizeof(iArr) / sizeof(*iArr)]`
- `mval` – a reference variable to store the maximum element of the array, which is `mVal`.

And it returns the index of the maximum element. If there is more than one maximum element, then it returns the location of the first occurrence. Therefore, options b) and d) are the correct options.

Question 5

Consider the following code segment. *[MCQ, Marks 1]*

```
#include <iostream>
#include <iomanip>
using namespace std;
int main () {
    cout << setprecision(5) << setfill('0') << setw(9) << 20/3.0;
    return 0;
}
```

What will be the output?

a) `00006.66667`

b) `0006.6667`

c) `0000000006.66667`

d) `0000000006.70000`

Answer: b)
Explanation:
`20/3.0 = 6.6666...`
`setprecision(5)` makes it to `6.6667`
`setfill('0')` and `setw(9)` make it to `0006.6667`.

Question 6

Consider the following code segment (in C++11). *[MCQ, Marks 2]*

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <list>

int main() {
    std::list<int> li= { 1, 2, 3, 4, 5 };
    std::vector<int> vi(li.size());
    std::list<int>::iterator it1 = li.begin();
    std::vector<int>::iterator it2 = vi.begin();
    for(int i = 0; i < 3; i++){    //LINE-1
        it1++; it2++;
    }

    copy(it1, li.end(), it2);     copy (it1) to (i.end-1) at position (it2)

    for(it2 = vi.begin(); it2 != vi.end(); ++it2)
        std::cout << *it2;
    return 0;
}
```

What will be the output?

a) `12345`

b) `45000`

c) `00045`

d) `00123`

Answer: c)
Explanation:
The `for` loop at `LINE-1` sets `it1` and `it2` at the third position of `li` and `vi`, respectively. Therefore, the `copy` function copies the last two elements from `li` to the last two positions of `vi`.

Question 7

Consider the following program (in C++11) to compute the inner product between the integers in a vector `vi` and a list `li`. *[MSQ, Marks 2]*

```
#include <iostream>
#include <list>
#include <vector>
#include <numeric>

int operation1(int i, int j){ return i + j; }
int operation2(int i, int j){ return i * j; }

int main() {
    std::vector<int> vi  { 1, 2, 3 };
    std::list<int> li  { 10, 20, 30 };

    int n = inner_product(_____________________________________);    //LINE-1
    std::cout << n;
    return 0;
}
```

Choose the correct option to fill in the blank at `LINE-1` so that output becomes `140`.

a) `vi.begin(), vi.end(), li.begin(), 0, operation1, operation2`

b) `li.begin(), li.end(), vi.begin(), 0, operation1, operation2`

c) `li.begin(), li.end(), vi.begin(), 0, operation2, operation1`

d) `vi.begin(), vi.end(), li.begin(), 0, operation2, operation1`

Answer: a), b)
Explanation:
The implementation of `inner_product` function is similar to:

```
template<class In, class In2, class T, class BinOp, class BinOp2 >
T inner_product(In first, In last, In2 first2, T init, BinOp op1, BinOp2 op2) {
    while(first!=last) {
        init = op1(init, op2(*first, *first2));
        ++first; ++first2;
    }
    return init;
}
```

Thus, a) and b) both are the correct options.

Question 8

Consider the following code segment (in C++11). *[MCQ, Marks 2]*

```
#include<iostream>
#include<list>

struct divide{
    int d_;
    divide(int d = 1) : d_(d) { }
    bool operator()(int i){ return (i % d_ == 0); }
};


template<class T, class P>
T find_if(T first, T last, P pred) {
    while (_____________________________) ++first;    //LINE-1
        return first;
}

void show(std::list<int> li, int d){
    divide divi(d);
    std::list<int>::iterator it = find_if(li.begin(), li.end(), divi);    //LINE-3
    while(it != li.end()){
        std::cout << *it << " ";
        it = find_if(++it, li.end(), divi);
    }
}

int main(){
    std::list<int> li {7, 8, 1, 4, 2, 5, 6, 3};
    int d;
    show(li, 4);
    return 0;
}
```

Choose the appropriate option to fill in the blank at `LINE-1` so that it prints the values from list `li`, which are divisible by 4. So the output should be `8 4`

a) `first != last || !pred(*first)`

b) `first != last && !pred(*first)`

c) `first != last && pred(*first)`

d) `first != last || pred(first)`

Answer: b)
Explanation:
At `LINE-1`, the `while` loop condition must be:
`first != last && !pred(*first)`
such that the loop stops either when it reaches the end of the list or when the given predicate is satisfied.

Question 9

Consider the following code segment (in C++11). *[MCQ, Marks 2]*

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

template<typename Itr, typename T>
------------------------------------ {    //LINE-1
    int minpos = 0, i = 0;
    mval = *first++;
    while (first != last) {
        if(*first < mval){
            mval = *first;
            minpos = i + 1;
        }
        ++first;
        ++i;
    }
    return minpos;
}

int main(){
    std::vector<int> vi = { 3, 2, 1, 1, 6, 8, 7};
    int mVal = 0;
    int pos = minimum(vi.begin(), vi.end(), mVal);
    std::cout << pos << " : " << mVal;
    return 0;
}
```

Identify the correct function header(s) for `minimum` to fill in the blank at `LINE-1` such that the program finds out the minimum element of the vector `vi` and the output is `2 : 1`.

a) `int minimum(Itr first, Itr last, T mval)`

b) `int minimum(Itr first, Itr last, T& mval)`

c) `int minimum(T first, T last, Itr& mval)`

d) `int minimum(Itr first, T last, T& mval)`

Answer: b), c)
Explanation:
The actual parameters of the function `minimum` are as follows:

- `vi.begin()` and `vi.end()`: Return iterators pointing to the first element and last element in the vector.
- `mval`: Must be pass-by-reference to store the minimum element of the vector.

And it returns the index of the minimum element. Therefore, option b) and c) are the correct options.

Programming Questions

Question 1

Consider the following program.

- Fill in the blank at `LINE-1` to inherit from `unary_function`.
- Fill in the blank at `LINE-2` with appropriate initializer list.
- Fill in the blank at `LINE-3` to override function call operator.

The program must satisfy the sample input and output. *Marks: 3*

```
#include<iostream>
#include<algorithm>
#include<vector>
struct Calculate : ____________________________________ { //LINE-1
    int inc;
    double sum;
    Calculate() : _________________ {} //LINE-2
    ___________________________________________ //LINE-3
};
int main(){
    std::vector<double> vd;
    int n;
    double d;
    std::cin >> n;
    for(int i = 0; i < n; i++){
        std::cin >> d;
        vd.push_back(d);
    }
    Calculate com = for_each(vd.begin(), vd.end(), Calculate());
    std::cout << "avg = " << com.sum / com.inc;
    return 0;
}
```

Public 1

```
Input: 5 1.2 2.3 3.4 4.5 5.6
Output: avg = 3.4
```

Public 2

```
Input: 6 4.3 -9 -1.3 5.4 2.3 4.3
Output: avg = 1
```

Private

```
Input: 7 1 2 3 4 5 6 8
Output: avg = 4.14286
```

Answer:

```
LINE-1:  public std::unary_function<double, void>
LINE-2:  inc(0), sum(0.0)
```

`LINE-3: void operator() (double x) { sum += x; ++inc; }`

Explanation:

The functor `Calculate` inherits the `unary_function` as:

`struct Calculate : public std::unary_function<double, void>`

,since it accepts an int as input as `double` without any return.

The constructor must initialize `sum = 0.0` and `inc = 0`. Thus, it must be:

`Calculate() : inc(0), sum(0.0) {}`

The function call operator must be overloaded to find the count of the elements of the array, and to get the sum of the elements. Thus, it must be:

`void operator() (double x) { sum += x; ++inc; }`

Question 2

Consider the following program, which takes 10 numbers as input, and computes the frequency of each number. Fill in the blanks as per the instructions given below:

- at `LINE-1` with appropriate statement to iterate over the given list `li`,
- at `LINE-2` with appropriate statement to iterate over the given map `fq`,

such that it will satisfy the given test cases. *Marks: 3*

```
#include <iostream>
#include <map>
#include <list>

std::map<int, int> Frequency(std::list<int> li){
    std::map<int, int> fq;
    for (_____________________________________)   //LINE-1
        fq[*it]++;
    return fq;
}

void print(std::map<int, int> fq){
    for (_____________________________________)    //LINE-2
        std::cout << it->first << " => " << it->second << ", ";
}

int main() {
    std::list<int> li;
    int a;
    for(int i = 0; i < 10; i++){
        std::cin >> a;
        li.push_back(a);
    }
    std::map<int, int> fq = Frequency(li);
    print(fq);
    return 0;
}
```

Public 1

```
Input: 10 30 50 20 30 40 50 10 20 10
Output: 10 => 3, 20 => 2, 30 => 2, 40 => 1, 50 => 2,
```

Public 2

```
Input: 10 20 10 30 10 40 10 50 20 30
Output: 10 => 4, 20 => 2, 30 => 2, 40 => 1, 50 => 1,
```

Private

```
Input: 10 20 30 40 50 40 20 30 10 10
Output: 10 => 3, 20 => 2, 30 => 2, 40 => 2, 50 => 1,
```

Answer:

```
LINE-1:  std::list<int>::iterator it = li.begin(); it != li.end(); ++it
LINE-2:  std::map<int, int>::iterator it = fq.begin(); it != fq.end(); ++it
```

Explanation:

The `std::list<int>::iterator` type can be used to iterate through the given list `li`, so the statement at `LINE-1` ca be:

```
for (std::list<int>::iterator it = li.begin(); it != li.end(); ++it)
```

Similarly, the `std::map<int, int>::iterator` type can be used to iterate through the given map `fq`, so the statement at `LINE-2` can be:

```
for (std::map<int, int>::iterator it = fq.begin(); it != fq.end(); ++it)
```

Question 3

Consider the following program that filters the elements from a given list `li` within a given upper and lower bounds and prints them.

- Fill in the blanks at `LINE-1` with appropriate template declaration.
- Fill in the blanks at `LINE-2` with an appropriate condition of the `while` loop.
- Fill in the blanks at `LINE-3` with an appropriate call to function `find` such that it returns an iterator object to the first element from `li` that match the given predicate with upper and lower bounds.

The program must satisfy the sample input and output. *Marks: 3*

```
#include<iostream>
#include<list>
template<typename T>
struct boundIt{
    T ub_, lb_;
    boundIt(T lb = 0, T ub = 0) : ub_(ub), lb_(lb) { }
    bool operator()(T x){ return (x <= ub_ && x >= lb_); }
};
_____________________________________ //LINE-1
T find(T first, T last, Pred prd) {
    while (_____________________________) ++first; //LINE-2
    return first;
}
void display(std::list<int> li, int lb, int ub){
    boundIt<int> bnd(lb, ub);
    ____________________________________________________; //LINE-3
    while(it != li.end()){
        std::cout << *it << " ";
        it = find(++it, li.end(), bnd);
    }
}
int main(){
    std::list<int> li {30, 70, 10, 40, 20, 50, 60, 80};
    int l, u;
    std::cin >> l >> u;
    display(li, l, u);
    return 0;
}
```

Public 1

```
Input: 30 55
Output: 30 40 50
```

Public 2

```
Input: 20 80
Output: 30 70 40 20 50 60 80
```

Private

```
Input: 50 88
Output: 70 60 80
```

Answer:

```
LINE-1:  template<typename T, typename Pred>
```

or

```
LINE-1:  template<class T, class Pred>
LINE-2:  first != last && !prd(*first)
LINE-3:  std::list<int>::iterator it = find(li.begin(), li.end(), bnd)
```

Explanation:

At `LINE-1`, the function the appropriate funcition template is:

```
template<typename T, typename Pred>
```

or

```
template<class T, class Pred>
```

At `LINE-2`, the `while` loop condition must be:

```
first != last && !prd(*first)
```

At `LINE-3`, the first element matches the predicate can be found as:

```
std::list<int>::iterator it = find(li.begin(), li.end(), bnd)
```

Programming in Modern C++: Assignment Week 10

Total Marks : 27

Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur, Kharagpur – 721302
partha.p.das@gmail.com

September 20, 2024

Question 1

Consider the program (in C++11) given below. *[MCQ, Marks 1]*

```
#include <iostream>

__________________ {            // LINE-1
    double dct = 0.05;
    double getVal(double pri){
        return pri * dct;
    }
}

________________________ {            // LINE-2
    double dct = 0.07;
    template<typename T>
    T getVal(T pri){
        return pri * dct;
    }
}

int main(){
   std::cout << ns1::getVal(105.0) << " ";
   std::cout << ns2::getVal(105) << " " << getVal(105.0);
   return 0;
}
```

Choose the appropriate option to fill in the blanks at `LINE-1` and `LINE-2` so that the output becomes
`5.25 7 7.35`

a) `LINE-1: namespace ns1`
   `LINE-2: namespace ns2`

b) `LINE-1: namespace ns1`
   `LINE-2: inline namespace ns2`

c) `LINE-1: inline namespace ns1`
   `LINE-2: namespace ns2`

d) `LINE-1: inline namespace ns1`
`LINE-2: inline namespace ns2`

Answer: b)
Explanation:
As per the output of `ns1::getVal(105.0)` and `ns2::getVal(105)`, the `ns1` and `ns2` must have basic namespace definition. However, since `getVal(105.0)` invoke the function `getVal` from `ns2`, `ns2` must be the default namespace. Thus, at `LINE-1` and `LINE-2`, we must have:

```
namespace ns1
inline namespace ns2
```

Question 2

Consider the program (in C++11) given below. *[MCQ, Marks 1]*

```
#include <iostream>
int main( ){
    int a = 1;
    const int b = 1;
    int& i1 = a;
    const int& i2 = b;
    auto x1 = i1;
    auto x2 = i2; //LINE-1
    std::cout << ++x1 << " " << ++x2 << " "; //LINE-2
    std::cout << i1 << " " << i2;
    return 0;
}
```

What will be the output/error?

a) `Compiler error at LINE-1: auto connot deduce to cv-qualifier`

b) `Compiler error at LINE-2: read-only x2 cannot be modified`

c) `2 2 2 2`

d) `2 2 1 1`

Answer: d)
Explanation:
Since `auto` never deduces adornments like cv-qualifer or reference (however, no error or exception is generated), the inferred type of `x1` and `x2` is `int`. Thus, the changes in `x1` and `x2` are not reflected on `i1` and `i2`.

Question 3

Consider the following code segment (in C++11). *[MSQ, Marks 2]*

```
#include <iostream>
#include <vector>
void change(std::vector<int>& iv){
    auto j = 2;
    for(_________________ : iv) //LINE-1
        i *= j;
}
int main( ){
    std::vector<int> iVec { 10, 20, 30, 40 };
    change(iVec);
    for(auto i : iVec)
        std::cout << i << ", ";
    return 0;
}
```

Choose the appropriate option/s to fill in the blank at `LINE-1` such that the output is `20, 40, 60, 80,`

a) `auto i`

b) `decltype(j) i`

c) `decltype((j)) i`

d) `decltype(iv[j]) i`

Answer: c), d)
Explanation:
In options a) and b), the inferred type of `i` is `int`. Thus, the changes made in `i` are not reflected in the vector `iVec`. So, the O/P is `20, 40, 60, 80,`
In options c) and d), the inferred type of `i` is `int&`. Thus, the changes made in `i` are reflected in the vector `iVec`.

Question 4

Consider the following code segment (in C++14). *[MSQ, Marks 2]*

```
#include<iostream>

struct LHS{
    int i {10};
    int operator()() { return i ; }
};

struct RHS{
    int i {10};
    int& operator()() { return i ; }
};


template < typename T >
---------------------------------------- {    //LINE-1
    return rf() ;
}

int main(){
    LHS f1;
    RHS f2;
    std::cout << caller(f1) << " ";
    std::cout << (caller(f2) = 100);
    return 0;
}
```

Choose the appropriate option/s to fill in the blank at `LINE-1` such that the output is `10 100`.

a) `auto caller( T& rf )`

b) `auto caller( T& rf ) -> decltype(rf())`

c) `int& caller( T& rf )`

d) `decltype(auto) caller( T& rf )`

Answer: b), d)
Explanation:
The call `caller(f1)` evaluates to prvalue of type `int`.
The call `caller(f2) = 100` evaluates to lvalue of type `int&`.
Since plain `auto` never deduces to a reference, option a) fails for lvalue.
Since plain `int&` always deduces to a reference, option c) fails for prvalue.
Option b) and d) works for `lvalue` as well as `prvalue`. Thus these two are correct options.

Question 5

Consider the following code segment (in C++11). *[MCQ, Marks 2]*

```
#include <iostream>

int main( ){
    int n = 10;

    int& a = n;
    const int& b = 10;

    auto x1 = a;
    auto x2 = b;
    decltype(a) x3 = a;
    decltype(b) x4 = a;

    ++x1;    //LINE-1
    ++x2;    //LINE-2
    ++x3;    //LINE-3
    ++x4;    //LINE-4

    return 0;
}
```

Which of the following line/s generate/s compilation error/s?

a) `LINE-1`

b) `LINE-2`

c) `LINE-3`

d) `LINE-4`

Answer: d)
Explanation:
Since `auto` never deduces adornments like cv-qualifier or reference (however, no error or exception is generated), the inferred type of `x1` and `x2` is `int`. For `x3`, the inferred type is `int&`, whereas for `x4`, the inferred type is `const int&`. Therefore, d) is the correct option.

Question 6

Consider the following code segment (in C++11). *[MSQ, Marks 2]*

```
#include<iostream>
#include<iomanip>

long double operator"" _M(long double x) {
    return x * 100;
}
long double operator"" _CM(long double x) {
    return x;
}
class distance{
    public:
        distance(int d1, int d2) : d1_(d1), d2_(d2){}
        int getDistance(){ return d1_ + d2_; }
    private:
        int d1_, d2_;
};
int main() {
    distance d(____________________);    //LINE-1
    std::cout << d.getDistance() << "CM";
    return 0;
}
```

Choose the appropriate option to fill in the blank at `LINE-1` such that the output is `1019CM`.

a) `10.0M, 19.0CM`

b) `10.0_M, 19.0_CM`

c) `(M)10.0, (CM)19.0`

d) `10_M, 19_CM`

Answer: b)
Explanation:
For user-defined numeric literal operators, the correct way to invoke them is to write them as `10.0_M, 19.0_CM`.
All other options are compilation error. Even option d) is wrong as numeric literal operators require exact type matching.
Intentionally kept as MSQ

Question 7

Consider the following program (in C++11). *[MCQ, Marks 2]*

```
#include <iostream>
#include <vector>
#include <initializer_list>

template<typename T>
class Number{
    public:
        Number() { std::cout << "1" << " "; }
        Number(int n) { std::cout << "2" << " "; }
        Number(std::initializer_list<int> elems) { std::cout << "3" << " "; }
        Number(int n, std::initializer_list<int> elms) { std::cout << "4" << " ";
    }
};
int main(){
    Number<int> n1(10);
    Number<int> n2({10, 20, 30});
    Number<int> n3{10, 20, 30};
    Number<int> n4 = {10, 20, 30};
    Number<int> n5(10, {10, 20, 30});
    return 0;
}
```

What will be the output?

a) `2 3 3 1 4`

b) `2 3 3 3 4`

c) `2 3 2 1 3`

d) `2 4 3 3 4`

Answer: b)
Explanation:
`Number<int> n1(10);` invokes parameterized constructor `Number(int n) { ... }`.
`Number<int> n2({10, 20, 30});`, `Number<int> n3{10, 20, 30};` and `Number<int> n4 = {10, 20, 30};` invoke the initializer list constructor `Number(initializer_list<int> elms){ ... }`.
`Number<int> n5(10, {10, 20, 30});` invokes the mixed constructor
`Number(int n, initializer_list<int> elms){ ... }`.

Question 8

Consider the following code segment (in C++11). *[MSQ, Marks 2]*

```
#include<iostream>

constexpr int f2(const int i){
    return i + 10;
}

void function(const int i){
    constexpr int n = 20;
    constexpr int c1 = n + 30;      //LINE-1
    constexpr int c2 = n + c1;      //LINE-2
    constexpr int c3 = n + i;       //LINE-3
    constexpr int c4 = n + f2(i);   //LINE-4
}

int main(){
    function(50);
    return 0;
}
```

Identify the line/s that generate/s compilation error/s.

a) `LINE-1`

b) `LINE-2`

c) `LINE-3`

d) `LINE-4`

Answer: c), d)
Explanation:
`constexpr` needs compile-time constant.
At `LINE-1`, c1 = n + 30; where `n` is a `constexpr` and `30` is a literal. Therefore, `c1` is a `constexpr`.
At `LINE-2`, c2 = n + c1 where `n` and `c1` both are `constexpr`. Therefore, `c2` is a `constexpr`.
At `LINE-3`, c3 = n + i; where `n` is a `constexpr`; however `i` is not a compile-time constant. Therefore, `c2` cannot be `constexpr`.
At `LINE-4`, c4 = n + f2(i); where `n` is a `constexpr`; however the call `f2(i)` fails since `i` is not a compile-time constant.

Question 9

Consider the following code segment (in C++11). *[MSQ, Marks 2]*

```
#include <iostream>

void show(char* str){ /*some code*/ }

template<typename FUNC, typename PARA>
void function(FUNC f, PARA p){
    f(p);
}

int main(){
    char s[15] = "Modern C++";
    function(show, s);          //LINE-1
    function(show, 0);          //LINE-2
    function(show, s[4]);       //LINE-3
    function(show, nullptr);    //LINE-4
    return 0;
}
```

Choose the call/s to `function` that will result in compiler error/s.

a) `LINE-1`

b) `LINE-2`

c) `LINE-3`

d) `LINE-4`

Answer: b), c)
Explanation:
For the call in `LINE-1`, the template type parameter `PARA` is deduced to `char*`. Thus, it does not generate any compiler error.
For the call in `LINE-2`, the template type parameter `PARA` is deduced to `int`. Thus, it generates a compiler error.
For the call in `LINE-3`, the template type parameter `PARA` is deduced to `char`. Thus, it generates a compiler error.
For the call in `LINE-4`, the template type parameter `PARA` is deduced to `std::nullptr_t` and the call `show(f, std::nullptr_t)` is syntactically correct.

Programming Questions

Question 1

Consider the following program (in C++11). Fill in the blanks as per the instructions given below:

- Fill in the blanks at `LINE-1` and `LINE-2` with appropriate headers for the definition of function `getValue()` belongs to class `G` and class `KG`.
- Fill in the blank at `LINE-3` with appropriate template definition.
- Fill in the blank at `LINE-4` with appropriate header for function `convert_weight`.

The program must satisfy the sample input and output. *Marks: 3*

```
#include <iostream>

class KG;
class G{
    public:
        G(double w) : w_(w) { }
        KG getValue();
        void print(){ std::cout << w_ << "G "; }
    private:
        double w_;
};

class KG{
    public:
        KG(double w) : w_(w) { }
        G getValue();
        void print(){ std::cout << w_ << "KG "; }
    private:
        double w_;
};

______________________ { return KG(w_ / 1000); }    //LINE-1
______________________ { return G(w_ * 1000); }     //LINE-2
______________________                              //LINE-3
___________________________________ {               //LINE-4
    return w.getValue();
}

int main(){
    double  a, b;
    std::cin >> a >> b;
    G o1(a);
    KG o2(b);
    convert_weight(o1).print();
    convert_weight(o2).print();
    return 0;
}
```

Public 1

```
Input: 10 10
Output: 0.01KG 10000G
```

Public 2

```
Input: 2000 20
Output: 2KG 20000G
```

Private

```
Input: 50 500
Output: 0.05KG 500000G
```

Answer:

```
LINE-1:  KG G::getValue()
LINE-2:  G KG::getValue()
LINE-3:  template <typename T>
```

Or

```
LINE-3:  template <class T>
LINE-4:  auto convert_weight(T w) -> decltype(w.getValue())
```

Or

```
LINE-4:  decltype(auto) convert_weight(T w)
```

Explanation:
At `LINE-1` and `LINE-2` the header for `getValue()` function can be written as:
`KG G::getValue()`
and
`G KG::getValue()`
At `LINE-3` the template, and at `LINE-4` the header for function `convert_weight` can be written as:
`template <typename T>` or `template <class T>`
and
`auto convert_weight(T w) -> decltype(w.getValue())` in C++11/14
or
`decltype(auto) convert_weight(T w)` in C++14

Question 2

Consider the following program in C++11/14. Fill in the blanks as per the instructions given below:

- at `LINE-1` with appropriate header and initialization list for the copy constructor,
- at `LINE-2` with the appropriate header for copy assignment operator overload,
- at `LINE-3` with appropriate header and initialization list for the move constructor,
- at `LINE-4` with the appropriate header for move assignment operator overload,

such that it will satisfy the given test cases. *Marks: 3*

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

class Integer {
    public:
        Integer(){}
        Integer(int i) : ip_(new int(i)) { }
        ___________________________________ { }    // LINE-1: copy constructor
        _______________________________ {      // LINE-2: copy assignment
            if (this != &n) {
                delete ip_;
                ip_ = new int(*(n.ip_) * 10);
            }
            return *this;
        }
        ~Integer() { delete ip_; }
        ___________________________ { n.ip_ = nullptr; }  // LINE-3: move constructor
        _____________________________ {        // LINE-4: move assignment
            if (this != &d) {
                ip_ = d.ip_;
                d.ip_ = nullptr;
            }
            return *this;
        }
        void show(){
            if(ip_ == nullptr)
                std::cout << "moved : ";
            else
                std::cout << *ip_ << " : ";
        }
        private:
            int* ip_ {nullptr};
};

int main(){
    int a;
    std::cin >> a;
    Integer n1(a);
    Integer n2 = n1;
```

```
    Integer n3;
    n3 = n1;
    n1.show();
    n2.show();
    n3.show();

    Integer n4 = std::move(n1);
    Integer n5;
    n5 = std::move(n1);
    n1.show();
    n4.show();
    n5.show();
    return 0;
}
```

Public 1

```
Input: 5
Output: 5 : 25 : 50 : moved : 5 : moved :
```

Public 2

```
Input: -10
Output: -10 : -50 : -100 : moved : -10 : moved :
```

Private

```
Input: 1
Output: 1 : 5 : 10 : moved : 1 : moved :
```

Answer:

```
LINE-1:  Integer(const Integer& n) :  ip_(new int(*(n.ip_) * 5))
LINE-2:  Integer& operator=(const Integer& n)
LINE-3:  Integer(Integer&& n) :  ip_(n.ip_)
LINE-4:  Integer& operator=(Integer&& d)
```

Explanation:
As per the output specified, the header and initialization list for copy constructor at `LINE-1` is:
`Integer(const Integer& n) :  ip_(new int(*(n.ip_) * 10))`,
the header for copy assignment operator for copy assignment is:
`Integer& operator=(const Integer& n)`,
the header and initialization list for move constructor at `LINE-3` is:
`Integer(Integer&& n) :  ip_(n.ip_)`,
the header for move assignment at `LINE-4` is:
`Integer& operator=(Integer&& d)`.

Question 3

Consider the following program (in C++11). Fill in the blanks as per the instructions given below:

- Fill in the blank at `LINE-1` with an appropriate template definition.
- Fill in the blank at `LINE-2` to complete the header for function `inner_product`.
- Fill in the blank at `LINE-3` to define the new type `Tmp`.

The program must satisfy the sample input and output. *Marks: 3*

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

__________________________              //LINE-1
void inner_product(_________________________) {    //LINE-2
    _______________________________;    // LINE-3: define new type Tmp
    /* Don't edit the following part */
    Tmp sum = 0;

    for (int i=0; i < v1.size(); ++i) {
        sum += v1[i] * v2[i];
    }
    std::cout << sum << " ";
}

int main(){
    float a;
    int b;
    double c;
    std::vector<float> Vec1;
    std::vector<int> Vec2;
    for(int i = 0; i < 3; i++) {
        std::cin >> a;
        Vec1.push_back(a);
    }
    for(int i = 0; i < 3; i++) {
        std::cin >> b;
        Vec2.push_back(b);
    }

    inner_product(Vec1, Vec2);
    return 0;
}
```

Public 1

```
Input:
1.5 2.5 3.5
1 2 3
Output: 17
```

Public 2

```
Input:
5.5 3.0 4.5
11 12 13
Output: 155
```

Private

```
Input:
3.14 3.15 3.15
10 20 30
Output: 188.9
```

Answer:

```
LINE-1:  template<typename U, typename V>
```

Or

```
LINE-1:  template<class U, class V>
LINE-2:  std::vector<U>& v1, std::vector<V>& v2
LINE-3:  typedef decltype(v1[0] * v2[0]) Tmp
```

Explanation:

At `LINE-1`, the appropriate template can be defined as:
`template<typename U, typename V>`
or `template<class U, class V>`
At `LINE-2` the header for function `inner_product` can be written as:
`void inner_product(std::vector<U>& v1, std::vector<V>& v2)`
At `LINE-3` the new type `Tmp` can be created as:
`typedef decltype(v1[0] * v2[0]) Tmp`

Programming in Modern C++: Assignment Week 11

Total Marks : 27

Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur, Kharagpur – 721302
partha.p.das@gmail.com

September 27, 2024

Question 1

Consider the program (in C++11) given below. *[MSQ, Marks 2]*

```
#include <iostream>

class classA {
    public:
        classA(const int& n) { std::cout << "base - lvalue : " << n << std::endl; }
        classA(int&& n) { std::cout << "base - rvalue : " << n << std::endl; }
};

class classB {
    public:
        template<typename T, typename U>
        classB(T&& n1, U&& n2) : ____________________ { }    //LINE-1
    private:
        classA a1_, a2_;
};
int main(){
    int i = 1;
    classB b1(i, 2);
    return 0;
}
```

Choose the appropriate option to fill in the blank at `LINE-1` so that the output becomes
`base - lvalue :  1`
`base - rvalue :  2`

a) `a1_(n1), a2_(n2)`

b) `a1_(std::forward<T>(n1)), a2_(std::forward<U>(n2))`

c) `a1_(n1), a2_(std::forward<U>(n2))`

d) `a1_(std::forward<U>(n1)), a2_(n2)`

Answer: b), c)
Explanation:

From the output we can understand that for object `a1` the l-value version of the constructors and for object `a2` the r-value version of the constructors need to be called. Since at the constructor of class `classB`, `n1` and `n2` are received as universal reference types, `n1` can be forwarded to class `classA` constructor as `a1_(n1)` or as `a1_(std::forward<T>(n1))` but `n1` needs to be forwarded to class `classA` constructor as `a1_(std::forward<T>(n2))`

Question 2

Consider the code segment (in C++11) given below. *[MCQ, Marks 2]*

```
template<typename T>
class Test{
    public:
        void f1(T&& n);                //LINE-1
        template<typename U>
        void f2(U&& n);                //LINE-2
        void f3(Test&& n);         //LINE-3
        template<typename V>
        void f4(std::vector<V>&& n);  //LINE-4
};
```

Identify the line/s where `&&` indicates a universal reference.

a) `LINE-1`

b) `LINE-2`

c) `LINE-3`

d) `LINE-4`

Answer: b)
Explanation:
Note that `&&` usually indicates rvalue reference. `&&` indicates a universal reference only where type deduction takes place.
At `LINE-1`, no type deduction takes place during function call (the type deduction takes place during class instantiation), therefore `&&` at `LINE-1` is just a rvalue reference, not a universal reference.
At `LINE-2`, the template type parameter `U` requires type deduction. Thus, `&&` at `LINE-2` indicates a universal reference.
At `LINE-3`, requires no type deduction. Thus, `&&` at `LINE-3` indicates a rvalue reference.
At `LINE-4`, the template type parameter `V` requires type deduction. However, since the form of function parameter is not `V&&` (it in form `std::vector<V>&&`), it indicates only rvalue reference.

Question 3

Consider the following code segment (in C++11). *[MSQ, Marks 2]*

```
#include <iostream>
enum class CCType {A, B, C};
enum class GRADE {A, B, C};
enum SECTION {A, B, C};
bool isA(CCType col){
    if(col == CCType::A) //LINE-1
        return true;
    return false;
}
bool isB(CCType col){
    if(col == GRADE::BEST) //LINE-2
        return true;
    return false;
}
bool isC(CCType col){
    if(col == C) //LINE-3
        return true;
    return false;
}
int main() {
    if(isC(CCType::C) && isB(CCType::B) &&
            isA(CCType::A))
        std::cout << "true";
    return 0;
}
```

Identify the statement/s which are true for the given program.

a) It generates compiler error at `LINE-1`

b) It generates compiler error at `LINE-2`

c) It generates compiler error at `LINE-3`

d) It generates the output `true`

Answer: b), c)
Explanation:
The statement `if(col == CCType::A)` compares between two `CCType` type elements, which compiles successfully.
The statement `if(col == GRADE::BEST)` compares between `CCType` type with `GRADE` type, which are not type castable. Thus it generates error.
The statement `if(col == C)` compares between `CCType` type with `int`, which are not type castable. Thus it generates error.
Since the code generates compiler error, it will not produce any output.

Question 4

Consider the following code segment (in C++11). *[MCQ, Marks 2]*

```
#include <iostream>
#include <algorithm>
#include <vector>

int main() {
    std::vector<int> v {20, 40, 30, 10, 50};
    /* code-block-1 */
    sort(v.begin(), v.end(), compare());
    for(int i : v)
        std::cout << i << " ";
    return 0;
}
```

Identify the appropriate code block to be placed at `code-block-1` such that the output is `50 40 30 20 10 .`

a) `bool compare(int a, int b){ return a > b; }`

b)
```
struct compare{
    bool operator()(int a, int b){ return a > b; }
};
```

c)
```
class compare{
    bool operator()(int a, int b){ return a > b; }
};
```

d) `bool operator()(int a, int b){ return a > b; }`

Answer: b)
Explanation:
A functor can be local to a function; however, it is not possible for a function or class. Therefore, b) is the correct option.

Question 5

Consider the following code segment (in C++11). *[MSQ, Marks 2]*

```
#include <iostream>
template<typename T>
class Test{
    public:
        Test() = delete;
        Test(T _i) : i(_i){ }
        Test(const Test& ) = delete;
        Test(Test&& ) = default;
    private:
        T i;
};
int main(){
    Test<int> d1;                    //LINE-1
    Test<int> d2(5);                 //LINE-2
    Test<int> d3 = d2;               //LINE-3
    Test<int> d4 = std::move(d2);    //LINE-4
    return 0;
}
```

Which of the following line/s generate/s compiler error/s?

a) `LINE-1`

b) `LINE-2`

c) `LINE-3`

d) `LINE-4`

Answer: a), c)
Explanation:
Since the default constructor and the copy constructor of class `Data` are explicitly deleted. `LINE-1` and `LINE-3`m generate compiler errors.

Question 6

Consider the following code segment (in C++11). *[MCQ, Marks 2]*

```
#include <iostream>

class emp{
    public:
        explicit emp() : emp(0) { std::cout << "1 "; }    //LINE-1
        explicit emp(const int i) : emp(i, dSalary) { std::cout << "2 "; } //LINE-2
        explicit emp(const double s) : emp(0, s) { std::cout << "3 "; } //LINE-3
        emp(int i, double s) : id_{i}, salary_{s} { std::cout << "4 ";} //LINE-4
    private:
        int id_ { -1 };
        double salary_ { 0.0 };
        static constexpr double dSalary = 5500.00;
};

int main(){
    emp e1;
    return 0;
}
```

What will be the output?

a) `1`

b) `2 1`

c) `4 2 1`

d) `1 2 4`

Answer: c)
Explanation:
The instatiation of object as `emp e1;` causes the follwoing consecutive calls to the constructors:

1. constructor at `LINE-1`,
2. constructor at `LINE-2`,
3. constructor at `LINE-4`.

Since the constructors are executed in reverse order of the calls, the output would be `4 2 1` .

Question 7

Consider the following program (in C++14). *[MCQ, Marks 2]*

```
#include <iostream>

template<typename T> T test = T(2.54);
int main(){
    test<int> = 100;
    auto cm_2_m = [](auto(cm)) { return cm * test<decltype(cm)>; }(100);
    auto cm_2_in = [](auto(cm)) { return cm * test<decltype(cm)>; }(100.0);
    std::cout << cm_2_m << ", " << cm_2_in;
    return 0;
}
```

What will be the output?

a) `254, 254`

b) `10000, 10000`

c) `10000, 254`

d) `254, 10000`

Answer: c)
Explanation:
In the expression: `[](auto(cm)) return cm * test<decltype(cm)>; (100)` , the inferred type of `cm` is `int`. So, the result is `10000`.
In the expression: `[](auto(cm)) return cm * test<decltype(cm)>; (100.0)` , the inferred type of `cm` is `double`. So, the result is `254`.

Question 8

Consider the following λ expression (in C++11). *[MCQ, Marks 2]*

```
auto test = [d, &sum](std::list<int> l) {
    for(auto i : l)
        sum += i;
    return (double)sum / l.size();
};
```

Identify the most appropriate option that define the equivalent closure object for the above lambda function.

a)
```
struct test_struct {
      int d;
      int& sum;
      test_struct(int d_, int& sum_) : d(d_), sum(sum_) { }
      void operator()(std::list<int> l, double& avg) const {
          for(auto i : l)
              sum += i;
          avg = (double)sum / l.size();
      }
  };
  auto test = test_struct(d, sum);
```

b)
```
struct test_struct {
      int d;
      int sum;
      test_struct(int d_, int& sum_) : d(d_), sum(sum_) { }
      operator()(std::list<int> l) {
          for(auto i : l)
              sum += i;
          return (double)sum / l.size();
      }
  };
  auto test = test_struct(d, sum);
```

c)
```
struct test_struct {
      int d;
      int sum;
      test_struct(int d_, int& sum_) : d(d_), sum(sum_) { }
      double operator()(std::list<int> l) const {
          for(auto i : l)
              sum += i;
          return (double)sum / l.size();
      }
  };
  auto test = test_struct(d, sum);
```

d)
```
struct test_struct {
      int d;
      int& sum;
      test_struct(int d_, int& sum_) : d(d_), sum(sum_) { }
      double operator()(std::list<int> l) const {
```

```
        for(auto i : l)
            sum += i;
        return (double)sum / l.size();
    }
};
auto test = test_struct(d, sum);
```

Answer: d)
Explanation:
For a λ-expression, the compiler creates a functor class with:

- data members:
  - a value member each for each value capture (`interval`)
  - a reference member each for each reference capture (`result`)
- a constructor with the captured variables as parameters
  - a value parameter each for each value capture
  - a reference parameter each for each reference capture
- a public inline const function call `operator()` with the parameters of the lambda as parameters, generated from the body of the lambda
- copy constructor, copy assignment operator, and destructor

Question 9

Consider the following code segment (in C++11). *[MSQ, Marks 2]*

```
#include <iostream>
#include <string>

class intClass {
    public:
        intClass(bool data) : d1_(data) {  }
        intClass(std::string data) : d2_(data) {  }

        explicit operator bool const() {
            if(d2_ == "")
                return d1_ == false ? 0 : 1;
            else
                return d2_ == "true" ? 1 : d2_ == "false" ? 0 : -1;
        }

    private:
        bool d1_ { false };
        std::string d2_ { "" };
};

int main(){
    std::string v = "false";
    intClass i1(true);
    intClass i2(v);
    std::cout << i1 << " "; //LINE-1
    std::cout << (bool)i2 << " "; //LINE-2
    std::cout << static_cast<bool>(i1) << " "; //LINE-3
    std::cout << bool(i2) << " "; //LINE-4
    return 0;
}
```

Which of the following lines will generate compiler error/s?

a) `LINE-1`

b) `LINE-2`

c) `LINE-3`

d) `LINE-4`

Answer: a)
Explanation:
Since the typecast (to `bool`) operator is overloaded at `LINE-1` is explicit, the typecast to `bool` must an explicit typecasting. `LINE-1` has an implicit which causes compiler error. The rest of the options are explicit typecasting.
Intentionally kept as MSQ

Programming Questions

Question 1

Consider the following program (in C++11).

- Fill in the blanks at `LINE-1` and `LINE-3` with appropriate template definitions.
- Fill in the blanks at `LINE-2` and `LINE-4` with appropriate parameters for `findMax` function.

The program must satisfy the sample input and output. *Marks: 3*

```
#include <iostream>

__________________________    //LINE-1
double findMin(________){ return num; }    //LINE-2

_____________________________________    //LINE-3
double findMin(_________________________){     //LINE-4
    return num <= findMin(nums...) ? num : findMin(nums...);
}

int main(){
    int a, b, c;
    double d, e, f;
    std::cin >> a >> b >> c;
    std::cin >> d >> e >> f;
    std::cout << findMin(a, b, c) << " ";
    std::cout << findMin(d, e, f) << " ";
    std::cout << findMin(a, b, c, d, e, f);
    return 0;
}
```

Public 1

```
Input:
10 20 30
12.5 15.5 20.5
Output:
10 10.5 10
```

Public 2

```
Input:
1 5 3
2.3 6.7 2.1
Output:
1 2.1 1
```

Private

```
Input:
400 30 100
10.65 100.56 200.43
```

```
Output:
30 10.65 10.65
```

Answer:

```
LINE-1:  template <typename T>
LINE-2:  T num
LINE-3:  template <typename T, typename...  Tail>
LINE-4:  T num, Tail...  nums
```

Explanation:

At `LINE-1`, the definition of the simple template is:

```
template <typename T>
```

, and at `LINE-3` the parameter of the function `findMin` are:

```
T num
```

At `LINE-3`, the definition of the variadic template is:

```
template <typename T, typename...  Tail>
```

, and at `LINE-4` the parameters of the function `findMin` are:

```
T num, Tail...  nums
```

Question 2

Consider the following program (in C++11). Fill in the blanks as per the instructions given below:

- Fill in the blank at `LINE-1` with an appropriate template declaration for the function `wrapper`.
- Fill in the blank at `LINE-2` with an appropriate header for function `wrapper`.
- Fill in the blank at `LINE-3` with an appropriate return statement for function `wrapper`.

The program must satisfy the sample input and output. *Marks: 3*

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

template<typename T, typename U>
struct Sum_op{
    double operator()(std::ostream& os, std::vector<T>&& v1, std::vector<U>&& v2){
        os << "rvalue version: ";
        double sum {0.0};
        for(T i : v1) sum += i;
        for(U i : v2) sum += i;
        return sum;
    }
    //template<typename T, typename U>
    double operator()(std::ostream& os, const std::vector<T>& v1,
                      const std::vector<U>& v2){
        os << "lvalue version: ";
        double sum {0.0};
        for(T i : v1) sum += i;
        for(U i : v2) sum += i;
        return sum;
    }
};

_____________________________________________    //LINE-1
_____________________________________________ {    //LINE-2
    ______________________________________;    //LINE-3
}

int main() {
    std::vector<int> iv;
    std::vector<double> dv;
    for(int i = 0; i < 3; i++){
        int a;
        std::cin >> a;
        iv.push_back(a);
    }
    for(int i = 0; i < 3; i++){
        double b;
        std::cin >> b;
        dv.push_back(b);
```

```
    }

    std::cout << wrapper(std::cout, Sum_op<int, double>(), iv, dv);
    std::cout << std::endl;
    std::cout << wrapper(std::cout, Sum_op<int, double>(), std::move(iv),
                    std::move(dv));
    return 0;
}
```

Public 1

```
Input:
10 20 30
1.5 2.5 2.5
Output:
lvalue version: 66.5
rvalue version: 66.5
```

Public 2

```
Input:
1 -3 -4
2.5 4.5 -3
Output:
lvalue version: -2
rvalue version: -2
```

Private

```
Input:
43 12 6
10.5 3.6 -98.3
Output:
lvalue version: -23.2
rvalue version: -23.2
```

Answer:

`LINE-1:` `template<typename F, typename... T>`

Or

`LINE-1:` `template<class F, class... T>`

`LINE-2:` `auto wrapper(std::ostream& os, F&& func, T&&... args) -> decltype(func(os, args...))` in C++11.

Or

`LINE-2:` `decltype(auto) wrapper(std::ostream& os, F&& func, T&&... args)` in C++14.

`LINE-3:` `return func(os, std::forward<T>(args)...)`

Explanation:

At `LINE-1` the template for function `wrapper` can be declared as:

`template<typename F, typename... T>`

Or

`template<class F, class... T>`

At `LINE-2` header for function `wrapper` can be written as:

`auto wrapper(std::ostream& os, F&& func, T&&... args) -> decltype(func(os, args...))` in C++11.

Or
`decltype(auto) wrapper(std::ostream& os, F&& func, T&&... args)` in C++14.
At `LINE-3` the return statement should be:
`return func(os, std::forward<T>(args)...)`

Question 3

Consider the following program (in C++11) to find factorials of 3 integers. Fill in the blanks as per the instructions given below:

- Fill in the blank at `LINE-1` to complete the lambda function for computing factorial.
- Fill in the blank at `LINE-2` to complete the lambda function for printing the vector `vec2`.

The program must satisfy the sample input and output. *Marks: 3*

```
#include <iostream>
#include <functional>
#include <vector>

int main() {
    std::vector<int> vec1;
    std::vector<long long> vec2;

    for(int i = 0; i < 3; i++){
        int a;
        std::cin >> a;
        vec1.push_back(a);
    }

    ____________________________________________________ {    //LINE-1
        return n > 1 ? n * factorial(n - 1) : 1;
    };

    for(auto i : vec1)
        vec2.push_back(factorial(i));

    ________________________________ {    //LINE-2
        for (auto i : x){ std::cout << i << " "; } }(vec2);

    return 0;
}
```

Public 1

```
Input: 2 3 4
Output: 2 6 24
```

Public 2

```
Input: 1 3 5
Output: 1 6 120
```

Private

```
Input: 2 4 6
Output: 2 24 720
```

Answer:

```
LINE-1:  const std::function<long long(int)> factorial = [&factorial](int n)
```

`LINE-2:  [](std::vector<long long> x)`

Explanation:

The lambda function for computing the factorial can be written as:

```
const std::function<long long(int)> factorial = [&factorial](int n) {
    return n > 1 ? n * factorial(n - 1) : 1;
};
```

The lambda function for printing the vector `vec2` can be written as:

```
[](std::vector<long long> x) { for (auto i : x){ std::cout << i << " "; } }(vec2);
```

Programming in Modern C++: Assignment Week 12

Total Marks : 27

Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur, Kharagpur – 721302
partha.p.das@gmail.com

October 4, 2024

Question 1

Which of the following types of smart pointers follow/s shared ownership policy? *[MSQ, Marks 1]*

a) `std::auto_ptr`

b) `std::std::shared_ptr`

c) `std::unique_ptr`

d) `std::weak_ptr`

Answer: b), d)
Explanation:
`std::auto_ptr` and `std::unique_ptr` are the two smart pointer types that support **exclusive ownership policy**.
`std::std::shared_ptr` and `std::weak_ptr` are the two smart pointer types that support **shared ownership policy**.

Question 2

Consider the program (in C++11) given below. *[MCQ, Marks 2]*

```
#include <iostream>

template <typename T>
class SPtr {
    public:
        explicit SPtr(T* ptr = nullptr) {
            std::cout << "SPtr::ctor(), ";
            ptr_ = ptr;
        }
        ~SPtr() { delete (ptr_); }
        T& operator*() {
            std::cout << "SPtr::operator*(), ";
            return *ptr_;
        }
        T* operator->() {
            std::cout << "SPtr::operator->(), ";
            return ptr_;
        }
    private:
        T* ptr_;
};

class Test{
    public:
        Test(){ std::cout << "Test::ctor(), "; }
        void fun(){ std::cout << "Test::fun(), "; }
};

int main(){
    SPtr<Test> sp(new Test());
    (*sp).fun();
    return 0;
}
```

What will be the output?

a) `SPtr::ctor(), Test::ctor(), Test::fun(), SPtr::operator->(),`

b) `Test::ctor(), SPtr::ctor(), SPtr::operator->(), Test::fun(),`

c) `SPtr::ctor(), Test::ctor(), SPtr::operator*(), Test::fun(),`

d) `Test::ctor(), SPtr::ctor(), SPtr::operator*(), Test::fun(),`

Answer: d)
Explanation:
The statement `SPtr<Test> sp(new Test());` calls the constructor of `Test` first and then calls the constructor of `SPtr`, which print `Test::ctor(), SPtr::ctor(),`.
The statement `(*sp).fun();` first calls the function `operator*()` from `SPtr` and then calls the function of `fun()` from `Test`, which print `SPtr::operator*(), Test::fun(),`.

Question 3

Consider the following code segment (int C++11). *[MCQ, Marks 1]*

```
#include <iostream>
#include <functional>

int func(int x1, int x2, int x3, const int& x4) {
    return x1 - (x2 + x3 + x4);
}
int main() {
    using namespace std::placeholders;
    int i = 5, j = 3;
    auto f = std::bind(func, _2, i, _1, std::cref(j));
    std::cout << f(1, 2, 3) << " ";
    i = j = -10;
    std::cout << f(1, 2, 3);
    return 0;
}
```

What will be the output?

a) `-7 6`

b) `-8 6`

c) `-7 7`

d) `-8 16`

Answer: a)
Explanation:
The call `f(100, 20, -100)` results in binding 1 to _1, 2 to _2, and 3 remains unused. The formal arguments for the first call to `func` are as `print(2, 5, 1, 3)`, which is evaluated as `-7`. The formal arguments for the second call to `func` are as `print(2, 5, 1, -10)` (since `j` is considered as reference type in `bind` function), which is evaluated as `6`.

Question 4

Consider the following code segment (in C++11). *[MCQ, Marks 2]*

```
#include<iostream>
#include<memory>

struct st_data{
    st_data() = default;
};

void use(const std::shared_ptr<st_data> p){
    auto p2 = p;
    std::cout << p.use_count() << " ";
}
int main(){
    st_data d;
    auto p1 = std::make_shared<st_data>(d);
    {
        std::shared_ptr<st_data> p2 = p1;
        std::cout << p1.use_count() << " ";
    }
    auto p2 = p1;
    std::cout << p1.use_count() << " ";
    use(p1);
    std::cout << p1.use_count() << " ";
    p1.reset();
    std::cout << p1.use_count();
    return 0;
}
```

What will be the output?

a) `2 2 3 2 2`

b) `2 2 4 2 0`

c) `2 3 4 3 0`

d) `2 3 4 2 2`

Answer: b)
Explanation:
The values of recsouce counter (RC) associated with the `shared_ptr` are explained in the code comments:

```
#include<iostream>
#include<memory>

struct st_data{
    st_data() = default;
};

void use(const std::shared_ptr<st_data> p){    //RC = 3
    auto p2 = p;                               //RC = 4
```

```
    std::cout << p.use_count() << " ";
}
int main(){
    st_data d;
    auto p1 = std::make_shared<st_data>(d);    //RC = 1
    {
        std::shared_ptr<st_data> p2 = p1;      //RC = 2
        std::cout << p1.use_count() << " ";
    }                                          //RC = 1
    auto p2 = p1;                              //RC = 2
    std::cout << p1.use_count() << " ";
    use(p1);
    std::cout << p1.use_count() << " ";     //RC = 2
    p1.reset();                                //RC = 0
    std::cout << p1.use_count();
    return 0;
}
```

Question 5

Consider the following code segment (in C++11). *[MCQ, Marks 2]*

```
#include <iostream>
#include <memory>

void update_share_ptrs(std::shared_ptr<int> sp1, std::shared_ptr<int>& sp2){
    sp1.reset(new int(0));
    sp2.reset(new int(0));
}

void testAndPrint(const std::weak_ptr<int>& wp){
    if(auto p = wp.lock())
        std::cout << *p << " ";
    else
        std::cout << "expired ";
}

int main() {
    auto p1 = std::make_shared<int>(1);
    auto p2 = p1;
    std::weak_ptr<int> wp1 = p1;
    std::weak_ptr<int> wp2 = p2;
    update_share_ptrs(p1, p2);
    testAndPrint(wp1);
    testAndPrint(wp2);

    update_share_ptrs(p2, p1);
    testAndPrint(wp1);
    testAndPrint(wp2);

    return 0;
}
```

What will be the output?

a) `1 1 expired expired`

b) `1 0 1 0`

c) `1 1 expired 1`

d) `1 1 1 expired`

Answer: a)
Explanation:
The function `update_ptr2` uses pass-by-value for first parameter and pass-by-reference for first parameter.
The statment `auto p2 = p1;` makes the RCs for `p1` and `p2` as 2.
The following statements:
`std::weak_ptr<int> wp1 = p1;`
`std::weak_ptr<int> wp2 = p2;`
makes the RCs for `wp1` and `wp2` as 2.
The call `update_ptrs(p1, p2);` resets the `p1` locally, however reset to `p2` (associated with

`wp2`) would be reflected in main (since it is pass-by-reference). Therefore, RCs for `p1` and `p2` become 1, and RCs for `wp1` and `wp2` also become 1. As a result, `Print(wp1)` and `Print(wp2)` print `1 1`.
The call `update_ptrs(p2, p1);` resets the `p2` locally, however reset to `p1` (associated with `wp1`) would be reflected in main (since it is pass-by-reference). Therefore, RCs for `p1` and `p2` become 1, while RCs for `wp1` and `wp2` become 0. As a result, `Print(wp1)` and `Print(wp2)` print `expired expired`.

Question 6

Consider the following code segment (in C++11). *[MCQ, Marks 2]*

```
#include <iostream>
#include <future>
#include <vector>

struct Calc_sum{
    Calc_sum(const std::vector<double> dv) : dv_(dv) { }
    double operator()() {
        double sum = 0.0;
        for(double i : dv_)
            sum += i;
        return sum;
    }
    std::vector<double> dv_;
};

double makeThreadedCall(const std::vector<double> dv){
    auto as = ______________________;    //LINE-1
    return as.get() ;
}

int main() {
    std::vector<double> arr {2.3, 4.5, 6.3, 2.3, 5.6, 3.5};
    std::cout << makeThreadedCall(arr) << std::endl;
    return 0;
}
```

Choose the appropriate option to fill in the blank at `LINE-1` such that output becomes `24.5`.

a) `std::thread(Calc_sum(dv))`

b) `std::async(Calc_sum(dv))`

c) `std::atomic(Calc_sum(dv))`

d) `std::thread{bind(Calc_sum(dv))}`

Answer: b)
Explanation:
Since `as.get()` must be waiting for fullfillment of the promise, which the return value of `Calc_sum`, the call at `LINE-1` must be `std::async(Calc_sum(dv))`.

Question 7

Consider program (in C++11) given below. *[MSQ, Marks 2]*

```
#include <iostream>
#include <vector>
#include <thread>
#include <chrono>

void addition(std::vector<int>& v){
    for(int i = 5; i < 9; i++)
        v.push_back(i);
}

int main(){
    std::vector<int> v;
    std::thread t1 { &addition, std::ref(v) };
    for(int i = 0; i < 4; i++)
        v.push_back(i);
    t1.join();
    for(int i : v)
        std::cout << i << " ";
    return 0;
}
```

What will NOT be the output of the program?

a) `0 1 2 3 5 6 7 8`

b) `5 6 7 8 0 1 2 3`

c) `5 0 6 1 7 2 8 3`

d) `0 1 2 3`

Answer: d)
Explanation:
The given program add elements to the vector concurrently, and the vector will be printed once all the elements are added. Therefore option a), b) and c) all can be valid output. However, d) cannot be an output.
Intentionally kept as MSQ

Question 8

Consider the following program (in C++11). *[MSQ, Marks 2]*

```
#include <iostream>
#include <functional>
#include <thread>
#include <mutex>

struct st_A{
    int A_count = 0;
};

struct st_B{
    int B_count = 0;
};

std::mutex A_mtx;
std::mutex B_mtx;

void request1(st_A& aObj, st_B& bObj, int an, int bn) {
    std::unique_lock<std::mutex> lck1(A_mtx);
    std::unique_lock<std::mutex> lck2(B_mtx);
    aObj.A_count += an;
    bObj.B_count += bn;
    std::cout << "R1: " << aObj.A_count << " " << bObj.B_count << std::endl;
}

void request2(st_A& aObj, st_B& bObj, int an, int bn) {
    std::unique_lock<std::mutex> lck2(B_mtx);
    std::unique_lock<std::mutex> lck1(A_mtx);
    aObj.A_count += an;
    bObj.B_count += bn;
    std::cout << "R2: " << aObj.A_count << " " << bObj.B_count << std::endl;
}

int main(){
    st_A rA;
    st_B rB;
    std::thread t1{ std::bind(request1, std::ref(rA), std::ref(rB), 10, 10) };
    std::thread t2{ std::bind(request2, std::ref(rA), std::ref(rB), 20, 20) };
    t1.join();
    t2.join();
    return 0;
}
```

Identify the statement/s that is/are **not true** about the program.

a) It generates output as:
   `R1:  10 10`
   `R2:  30 30`

b) It generates output as:
   `R1:  10 10`
   `R2:  20 20`

c) It generates output as:
`R2: 20 20`
`R1: 30 30`

d) It results in deadlock

Answer: b)
Explanation:
Since the code in `request1` and `request2` execute in a mutually exclusive manner, the output can be:
`R1: 10 10`
`R2: 30 30` or
`R2: 20 20 R1: 30 30`
However, it cannot be
`R1: 10 10`
`R2: 20 20` It may also happen that `t1` holds lock on `A_mtx`, and `t2` holds lock on `B_mtx`. Then, `t1` request to lock on `B_mtx`, and `t2` requests lock on `A_mtx`. It results in a deadlock. Therefore, b) is the correct option.
Intentionally kept as MSQ

Question 9

Consider the following code segment (in C++). *[MCQ, Marks 2]*

```
#include <iostream>
#include <thread>
#include <future>
#include <vector>

void change (std::promise<std::vector<int>>& pr, std::future<std::vector<int>>& fu)
{
    std::vector<int> v = fu.get();
    std::vector<int> res;
    long sum = 0;
    for(auto i : v)
        res.push_back(i * i);
    pr.set_value(res);
}

int main () {
    std::promise<std::vector<int>> p1;
    std::future<std::vector<int>> f1 = p1.get_future();

    std::promise<std::vector<int>> p2;
    std::future<std::vector<int>> f2 = p2.get_future();

    std::thread th {change, std::ref(p2), std::ref(f1)};

    std::vector<int> v { 10, 20, 30, 40 };

    ____________________;    //LINE-1
    ____________________;    //LINE-2
    for(auto i : rv)
        std::cout << i << " ";
    th.join();
    return 0;
}
```

Choose the appropriate option to fill in the blanks at `LINE-1` and `LINE-2` so that the output becomes `100 400 900 1600` .

a) `LINE-1: p1.set_value (v)`
   `LINE-2: auto rv = f1.get()`

b) `LINE-1: p1.set_value (v)`
   `LINE-2: auto rv = f2.get()`

c) `LINE-1: p2.set_value (v)`
   `LINE-2: auto rv = f1.get()`

d) `LINE-1: p2.set_value (v)`
   `LINE-2: auto rv = f2.get()`

Answer: b)
Explanation:

As per the following call:
`std::thread th {change, std::ref(p2), std::ref(f1)};` `p2` becomes the `promise` and `f1` becomes the `future`. Therefore the input value must be associated with `p1` and output must be extracted from `f2`. So, b) is the correct option.

Programming Questions

Question 1

Consider the program below (in C++11), which implements a smart pointer.

- Fill in the blank at `LINE-1` with appropriate header and initializer list for the copy constrcutor.
- Fill in the blank at `LINE-2` with appropriate header to overload dereferenceing operator.
- Fill in the blank at `LINE-3` with appropriate header to overload indirection operator.

The program must satisfy the given test cases. *Marks: 3*

```
#include<iostream>

class myData{
        int i_;
    public:
        myData() = default;
        explicit myData(int i) : i_(i) {}
        explicit myData(const myData& d) : i_(d.i_) {}
        void update(int i) { i_ = i; };
        friend std::ostream& operator<<(std::ostream& os, const myData& d){
            os << d.i_ << " ";
            return os;
        }
};

template <typename T>
class SmartPtr {
    public:
        explicit SmartPtr(T* pointee) : pointee_(pointee) { }
        ________________________________ {    //LINE-1: copy constructor
            other.pointee_ = nullptr;
        }
        ~SmartPtr() { if(pointee_ != nullptr) delete pointee_; }
        ________________________________ { return *pointee_; } // LINE-2: Dereferencing
        ________________________________ { return pointee_; }  // LINE-3: Indirection
    private:
        T* pointee_;
};

int main(){
    int a, b;
    std::cin >> a >> b;
    SmartPtr<myData> sp(new myData(a));
    std::cout << *sp;
    sp->update(b);
    std::cout << *sp;
    SmartPtr<myData> sp2 = sp;
    std::cout << *sp2;
    return 0;
}
```

Public 1

```
Input: 5 10
Output: 5 10 10
```

Public 2

```
Input: 10 20
Output: 10 20 20
```

Private

```
Input: 10 -10
Output: 10 -10 -10
```

Answer:

```
LINE-1:  SmartPtr(SmartPtr& other) :  pointee_(other.pointee_)
LINE-2:  T& operator*() const
LINE-3:  T* operator->() const
```

Explanation:

At `LINE-1`, the header and initializer list for the copy constructor can be written as:
`SmartPtr(SmartPtr& other) :  pointee_(other.pointee_)`
At `LINE-2`, the header of the function to overload the dereferencing operator can be written as: `T& operator*() const`
At `LINE-3`, the header of the function to overload the indirection operator can be written as:
`T* operator->() const`

Question 2

Consider the following program in C++14 to represent a doubly linked list (of generic type), which allows adding items at the front of the list. Complete the program as per the instructions given below.

- Fill in the blank at `LINE-1` with appropriate statements to traverse the list in forward direction.
- fill in the blank at `LINE-2` with appropriate statements to traverse the list in backward direction,

The program must satisfy the sample input and output. *Marks: 3*

```
#include<iostream>
#include<memory>

template<typename T>
class mylist;

template<typename U>
class mynode{
    public:
        mynode(U _info) : info(_info), next(nullptr) {}
        friend mylist<U>;
    private:
        U info;
        std::shared_ptr<mynode<U>> next;
        std::weak_ptr<mynode<U>> prev;
};

template<typename T>
class mylist{
    public:
        mylist() = default;
        void add(const T& item){
            std::shared_ptr<mynode<T>> n = std::make_shared<mynode<T>>(item);
            if(first == nullptr){
                first = n;
                last = first;
            }
            else{
                n->next = first;
                first->prev = n;
                first = n;
            }
        }
        void biTraverse(){
            for(____________________________________)   //LINE-1
                std::cout << t->info << " ";
            std::cout << std::endl;
            for(____________________________________)   //LINE-2
                std::cout << p->info << " ";
        }
```

```
    private:
        std::shared_ptr<mynode<T>> first = nullptr;
        std::shared_ptr<mynode<T>> last = nullptr;
};
int main(){
    mylist<int> il;
    int n, a;
    std::cin >> n;
    for(int i = 0; i < n; i++){
        std::cin >> a;
        il.add(a);
    }
    il.biTraverse();
    return 0;
}
```

Public 1

```
Input:
4
10 20 30 40
Output:
40 30 20 10
10 20 30 40
```

Public 2

```
Input:
5
1 2 3 4 5
Output:
5 4 3 2 1
1 2 3 4 5
```

Public 3

```
Input:
1
10
Output:
10
10
```

Private

```
Input:
3
-10 -20 30
Output:
30 -20 -10
-10 -20 30
```

Answer:

```
LINE-1:  std::shared_ptr<mynode<T>> t = first; t != nullptr; t = t->next
```

`LINE-2: std::weak_ptr<mynode<T>> t = last; auto p = t.lock(); t = p->prev`
Explanation:
Since in `class mynode`, `next` is a `shared_ptr`, the `for` loop for forward traversal must be:
`std::shared_ptr<mynode<T>> t = first; t != nullptr; t = t->next)`
Since in `class mynode`, `prev` is a `weak_ptr`, the `for` loop for reverse traversal must be:
`std::weak_ptr<mynode<T>> t = last; auto p = t.lock(); t = p->prev`
Note that `last` is a `shared_ptr` while `prev` is a `weak_ptr`. So we need to devise a way to navigate between the two. Recall that `weak_ptr` cannot be used to access the pointee. So we first get `weak_ptr t` from `last`. Now for access, we get a `shared_ptr p` by `lock`ing the `weak_ptr t`. This will used in the loop body. Finally, to progress backward, we get `p->prev` and keep in the `weak_ptr t`. That completes the solution.

Question 3

Consider the following program (in C++11).

- Fill the blanks at `LINE-1` and `LINE-3` by locking the mutex object.
- Fill the blanks at `LINE-2` and `LINE-4` by unlocking the mutex object.

The program must satisfy the sample input and output. *Marks: 3*

```
#include <iostream>
#include <thread>
#include <functional>
#include <chrono>
#include <mutex>
std::mutex ac_mtx;

class account {
    public:
        account() = default;
        void deposit(double amount){
            ______________________;    //LINE-1
            update_amount = amount;
            int delay = (int)((double)std::rand() / (double)(RAND_MAX)* 10);
            std::this_thread::sleep_for(std::chrono::milliseconds(delay));
            balance_ += update_amount;
            ______________________;    //LINE-2
        }
        void withdrawal(double amount){
            ______________________;    //LINE-3
            update_amount = amount;
            int delay = (int)((double)std::rand() / (double)(RAND_MAX)* 10);
            std::this_thread::sleep_for(std::chrono::milliseconds(delay));
            balance_ -= update_amount;
            ______________________;    //LINE-4
        }
        void show_balance() { std::cout << balance_; }
        private:
            double update_amount {0.0};
            double balance_ {0.0};
};

void sender(account& ac, int n){
    for(int i = 0; i < n; i++){
        int delay = (int)((double)std::rand() / (double)(RAND_MAX)* 10);
        std::this_thread::sleep_for(std::chrono::milliseconds(delay));
        ac.deposit((i + 1) * 10);
    }
}

void receiver(account& ac, int n){
    for(int i = n - 1; i >= 0; i--){
        int delay = (int)((double)std::rand() / (double)(RAND_MAX)* 30);
        std::this_thread::sleep_for(std::chrono::milliseconds(delay));
```

```
        ac.withdrawal((i + 1) * 10);
    }
}

int main(){
    int n, m;
    std::cin >> n >> m;
    account ac;
    std::thread t1{ std::bind(sender, std::ref(ac), n) };
    std::thread t2{ std::bind(receiver, std::ref(ac), m) };

    t1.join();
    t2.join();
    ac.show_balance();
    return 0;
}
```

Public 1

```
Input: 20 20
Output: 0
```

Public 2

```
Input: 10 20
Output: -1550
```

Private

```
Input: 20 10
Output: 1550
```

Answer:
`LINE-1:` `ac_mtx.lock()`
`LINE-2:` `ac_mtx.unlock()`
`LINE-3:` `ac_mtx.lock()`
`LINE-4:` `ac_mtx.unlock()`
Explanation:
At `LINE-1` and `LINE-3`, locking of the mutex object can be written as:
`ac_mtx.lock()`
At `LINE-2` and `LINE-4`, unlocking of the mutex object can be written as:
`ac_mtx.unlock()`