Programming in 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

August 23, 2020

Question 1

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

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

int main() {
    int a[3] = { 10, 20, 30 };
    int __________;    // LINE-1

    p = &a;
    cout << (*p)[0] << " " << (*p)[1] << endl;

    return 0;
}
```

Fill in the blank at LINE-1 with appropriate option/s such that the output is: `10 20`

a) `*p`

b) `**p`

c) `(*p)[3]`

d) `*p[3]`

Answer: c)
Explanation:
In statement `p = &a;`, `&a` is the base address of an array `a`, which consists of 3 integers. Hence, `int (*p)[3]` is the appropriate declaration at LINE-1, as it means `p` is a pointer to an array of 3 integers. So option c) is correct.
Note: This question is made MSQ intentionally

Question 2

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

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

int main() {
    int a[] = { 10, 20, 30, 40, 50, 60 }, *p;
    p = a + 5;                // LINE-1
    int i = 0;

    while (i < 3) {
        cout << p[-i] << " "; // LINE-2
        p++;
        i++;
    }

    return 0;
}
```

What will be the output?

a) `10 30 50`

b) `60 50 40`

c) `60 40 20`

d) `60 60 60`

Answer: d)
Explanation:
At LINE-1, `p = a + 5;` is nothing but `p = &a[5];`, so `p` points to the last element of array `a`. At LINE-2, `p[-i] = *(p - i)`, initially when `i = 0`, `p[-i] = *(p - i) = *p = p[5] = 60`. Again, in every step, both `p` and `i` get incremented by 1. Hence, LINE-1 will execute `*((P + 1) - (i + 1)) = *(p - i) = *p = p[5] = 60`. As the loop repeat for 3 times, the output will be `60 60 60`, i.e., option d).

Question 3

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

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

struct shape {
    int type;
    union g_shape {
        struct rect_share { int pt1, pt2; } r1;
        struct tri_shape { int pt1, pt2, pt3; } r2;
    };
    union g_shape s;
};
```

What will be the `sizeof(shape)` (consider the `sizeof(int) = 4`) ?

a) `8`

b) `12`

c) `16`

d) `24`

Answer: c)
Explanation:
`sizeof(shape) = sizeof(type) + sizeof(s) = 4 (as  sizeof(int) = 4) + sizeof(s)`.
Now `sizeof(s) = sizeof(union g_shape) = 12`. As for `union` the allocation take place for largest member, in this case that is `r2`.
Hence `sizeof(shape) = 4 + 12 = 16` bytes, i.e. option c).

Question 4

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

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

int main() {
    int a[] = { 5, 20, 15 };
    int *arr[3] = { a, a + 1, a + 2 };

    cout << *arr[*arr[1] - 19];

    return 0;
}
```

What will be the output?

a) `5`

b) `20`

c) `15`

d) `Unpredictable value`

Answer: b)
Explanation:
`arr[3]` is an array of 3 integer addresses which are {`a = &a[0], a + 1 = &a[1], a + 2 = &a[2]`}. Now let us evaluate the O/p statement
Step-1: `*arr[*arr[1] - 19];`. Here `*arr[1]` denotes value at the address arr[1] which is the value of `a[1] = 20`.
step-2: `*arr[20-19] = *arr[1]`.
Step-3: `*arr[1]` which is value of `a[1]` i.e. 20.

Question 5

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

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

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

int main() {
    int data[] = { 40, 30, 90, 10, 20, 50, 70, 60, 80 };

    sort(data, __________, compare);    // LINE-1

    for (int i = 0; i < 9; i++)
        cout << data[i] << " ";

    return 0;
}
```

Fill in the blank with appropriate option/s, such that the output is:
`90 40 30 20 10 50 70 60 80`

a) `data + 9`

b) `data + 5`

c) `data[8]`

d) `&data[5]`

Answer: b), d)
Explanation:
The array `data` is sorted upto 5th element, rest of the array remains unsorted. Hence the blank at LINE-1 must be filled with address of 6th element of the array which is the address of `data[5]`. The address of `data[5]` can be presented either by `&data[5]` or `data + 5`. Hence option b) and d) are correct.

Question 6

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

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

int main() {
    string str1 = "Physical ";
    string str2 = "Science";

    str1 = __________;    // LINE-1
    cout << str1;

    return 0;
}
```

Fill in the blank at LINE-1 such that the output is: `Physical Science`.

a) `str1 + str2`

b) `strcat(str1,str2)`

c) `strcat(strcpy(str2,str1),str2)`

d) `str1.append(str2)`

Answer: a), d)
Explanation:
`str1` and `str2` are two string type variables, so the concatenation is similar to add two string that is `str1 + str2`. Similarly `str2` can be appended at the end of `str1` using `str1.append(str2)`. So, option a) and d) are correct.
Note: *The time being we should remember that the two string can be concatenated using '+' operator, but its mechanism will be explained in the coming weeks as operator overloading*

Question 7

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

```
#include <iostream>
#include <vector>

int main() {
    std::vector<int> myvector(5);
    for (int i = 1; i < 5; i++)
        myvector[i] = i;        // LINE-1
    myvector.resize(3);         // LINE-2
    myvector.resize(4, 110);    // LINE-3
    myvector.resize(5);         // LINE-4

    for (int i = 0; i < myvector.size(); i++)
        std::cout << ' ' << myvector[i];

    return 0;
}
```

What will be the output?

a) `0 1 2 110 0`

b) `0 1 2 1 0`

c) `0 1 2 3 4`

d) `110 110 110 110 0`

Answer: a)
Explanation:
The `resize()` has two version of calling (i) `resize(n)` (ii) `resize(n, val)`. Here the function, resizes the container so that it contains `n` elements.
Case i) `resize(n)`: If n is smaller than the current container size, the content is reduced to its first n elements, removing those beyond (and destroying them).
Case ii) `resize(n, val)`: If n is greater than the current container size, the content is expanded by inserting at the end as many elements as needed to reach a size of `n`. If `val` is specified, the new elements are initialized as copies of `val`, otherwise, they are value-initialized. If n is also greater than the current container capacity, an automatic reallocation of the allocated storage space takes place and value of the reallocated storage become zero (that is static allocation).
Now, let's execute Line-1 to 4 and see the O/P in each step to understand whole mechanism.
After LINE-1, the elements of `myvector` are [0, 1, 2, 3, 4].
After LINE-2, the elements of `myvector` are [0, 1, 2].
After LINE-3, the elements of `myvector` are [0, 1, 2, 110].
After LINE-4, the elements of `myvector` are [0, 1, 2, 110, 0].
So, the `cout` gives the O/P as mentioned in a)

Question 8

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

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

union sample {
    int x;
    char y;
};

int main() {
    union sample ptr1;
    ptr1.x = 97;
    ptr1.y = 'B'; // ASCII Code of 'B' is 66

    union sample *ptr2 = &ptr1;

    cout << ptr2->x + (*ptr2).y;    // LINE-1

    return 0;
}
```

What will be the output/error?

a) `163`

b) `132`

c) `194`

d) `Compiler error: type mismatch for + operator`

Answer: b)
Explanation:
A union may have many members, but only one member can contain a value at any given time. Hence the latest value of 'B' which is 66 treated as the value of `x` and `y` both in `union sample`.
Now at LINE-1, `(*ptr2).y` is equivalent to `ptr2->y`. Hence the statement `ptr2->x + (*ptr2).y` = 66 + 66 = 132.

Question 9

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

```
int n = 10;

const int *p1 = &n;
int * const p2 = &n;
int const *p3 = &n;
int const * const p4 = &n;

*p1 = 20;    // STMT-1
*p2 = 20;    // STMT-2
*p3 = 20;    // STMT-3
*p4 = 20;    // STMT-4
```

Which statement / statements are correct?

a) `STMT-1`

b) `STMT-2`

c) `STMT-3`

d) `STMT-4`

Answer: b)
Explanation:
In statement `const int *p1 = &n;`, for `p1` the pointee is constant, hence `*p1` cannot be modified. So a) is incorrect.
In statement `int const *p3 = &n;`, again for `p3` the pointee is constant, hence `*p3` cannot be modified. So c) is incorrect.
In statement `int const * const p4 = &n;`, for `p4` the pointer and pointee both are constant, hence `*p4` cannot be modified. So d) is incorrect.
But in statement `int const *p2 = &n;`, for `p2` the pointer is constant, hence `*p2` can be modified. So b) is the correct option.
Note: This question is made MSQ intentionally

Programming Questions

Question 1

Consider the following code snippet. Fill in the blank at `LINE-1` to create a stack and at `LINE-2` to fill the stack with the value of array `str`.

Please check sample input and output. *Marks: 3*

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

int main() {
    char str[19] = "Programming";
    cin >> str ;
    __________;    // LINE-1

    for (int i = 0; i < strlen(str); i++)
        __________;    // LINE-2

    for (int i = 0; i < strlen(str) - 1; i++) {
        cout << s.top();
        s.pop();
    }

    return 0;
}
```

Public 1

```
Input: Programming
Output: gnimmargor
```

Public 2

```
Input: discover
Output: revocsi
```

Private

```
Input: Assignment
Output: tnemngiss
```

Answer:
LINE-1: `stack<char> s`
LINE-2: `s.push(str[i])`
Explanation:
As the stack has to work with `char` type. Hence at LINE-1, we need to declare it as `stack<char> s;`. At LINE-2, the elements can be pushed at stack as `s.push(str[i]);`.

Question 2

Consider the following program and fill the blank at LINE-1 to check if `s1` is not greater than `s2`. Look at the the sample input and output to understand the comparison operation and write proper return statements at LINE-2 and LINE-3. *Marks: 3*

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

bool StrCmp(string s1, string s2) {

    if (__________)    // LINE-1

        __________;    // LINE-2
    else
        __________;    // LINE-3
}

int main() {
    string str1, str2;

    cin >> str1 >> str2;

    cout << str1 << ", " << str2 << " : " << StrCmp(str1, str2);

    return 0;
}
```

Public 1

```
Input: hello sir
Output: hello, sir : 1
```

Public 2

```
Input: test done
Output: test, done : 0
```

Private

```
Input: hello student
Output: hello, student : 1
```

Answer:
Line-1: `s1 <= s2`
Line-2: `return true` or `return 1`
Line-3: `return false` or `return 0`
Explanation:
At LINE-1 the condition must be `if(s1 <= s2)`, then at LINE-2 it `return true;`, in LINE-3 it `return false;`.

Question 3

Consider the following program and fill in the blanks at LINE-1 to define a function pointer type `Fun_Ptr` such that it can be used to create a pointer to point `add()` function. At LINE-2 create function pointer `fp` to point to function `add()` such that it satisfies the given test cases.
Marks: 3

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

void add(int a, int b) {
    cout << a + b;
}

typedef __________; // LINE-1

Fun_Ptr __________; // LINE-2

int main() {
    int a, b;
    cin >> a >> b;

    (*fp)(a, b);

    return 0;
}
```

Public 1

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

Public 2

```
Input: 50 60
Output: 110
```

Private

```
Input: -40 30
Output: -10
```

Answer:
LINE-1: `void (*Fun_Ptr)(int, int)`
LINE-2: `fp = &add`
or
LINE-2: `fp = add`
Explanation:
At LINE-1, the type of function pointer, for `add()` function must be:
`void (*Fun_Ptr)(int, int);`
At LINE-2, the definition of function pointer to point to `add()` function must be:
`Fun_Ptr fp = &add;`
or
`Fun_Ptr fp = add`.

Question 4

Consider the following program and fill in the blank at LINE-1 to appropriate parameters of `rotate()` function such that it satisfies the test cases. *Marks: 3*

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

int main() {
    int data[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
    int n = sizeof(data) / sizeof(data[0]);
    int rotL;

    cin >> rotL;
    rotate(____________________);  // LINE-1

    for (int i = 0; i < n; i++)
        cout << data[i] << ' ';

    return 0;
}
```

Public 1

```
Input: 3
Output: 4 5 6 7 8 9 1 2 3
```

Public 2

```
Input: 5
Output: 6 7 8 9 1 2 3 4 5
```

Private

```
Input: 2
Output: 3 4 5 6 7 8 9 1 2
```

Answer:
LINE-1: `data, data + rotL, data + n`
or
LINE-1: `&data[0], &data[rotL], &data[n]`
Explanation:
`rotate(first, middle, last);`
`first, last` : are the pointing to the initial and final positions of the array to be rotated,
`middle` : pointing to the element within the range `[first, last]` that is moved to the first position in the range.
Hence, the LINE-1 has to be filled up with:
LINE-1: `data, data + rotL, data + n`
or
LINE-1: `&data[0], &data[rotL], &data[n]`