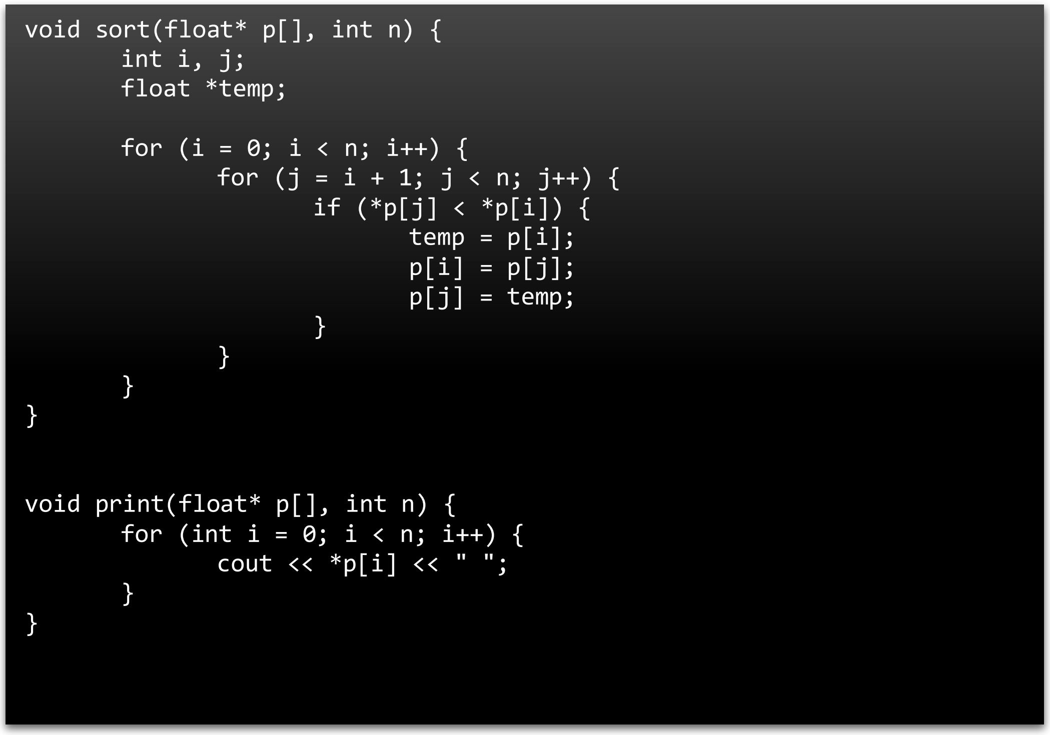
Name:	**Abdul Rehman**
Roll No:	**19L-1135**
Section:	**CS-D**
Course:	**Object Oriented Programming**
Submission: date	**10-05-2020**

Assignment #3

Q-1

(1)



```
void sort(float* p[], int n) {
	int i, j;
	float *temp;

	for (i = 0; i < n; i++) {
		for (j = i + 1; j < n; j++) {
			if (*p[j] < *p[i]) {
				temp = p[i];
				p[i] = p[j];
				p[j] = temp;
			}
		}
	}
}


void print(float* p[], int n) {
	for (int i = 0; i < n; i++) {
		cout << *p[i] << " ";
	}
}
```

(2)

CODE:

```
int x = 5;
int* y = new int(3);  ///Constuctor being called storing int 3 in *y.
int** z = &y;         ///z[][]=address of y
int A[5] = { 1,2,3,4,5 };
cout << *y;           ///it will cout the value that pointer"y" is pointing to.
cout << **z;          ///it will cout the value that pointer"z" is pointing to.
cout << *&x;          ///it will cout the value that the address of "x" is pointing to.
cout << A[4];         ///it will cout the A[4] element of array "A".
cout << *(A + 2);     ///it will cout the A[0+2] = A[2] element of array "A".
cout << *(A + *y);    ///it will cout the A[0+3] = A[3] element of array "A".
cout << A[**z];       ///it will cout the A[3] element of array "A".
cout << A[x];         ///it will cout garbage value as A[5] is out of the range of
array "A".
```

Output:

```
3
3
5
5
3
4
4
G
```

where “G” is for garbage value.

(3)

CODE:

```
int meaning = 42;
int *life = &meaning;
int **universe = &life;
int ***everything = &universe;
cout << ***everything << endl;
cout << life << endl;
cout << universe << endl;
cout << everything << endl;
delete life;
life = nullptr;
universe = nullptr;
everything = nullptr;
```

Answer:

As there is no use of the operator “new”, so all the variables will be made on stack. Therefore, the line”`delete life;`” will give error as “life” is a stack variable and we can’t delete stack variables using delete operator because it works only on heap variables.

Q-2

CODE:

```
int i = 0, j = 0, rows = 4, cols = 7;
int **Arr = new int *[rows];
for (i = 0; i < rows; ++i)
       Arr[i] = new int[cols - i];
int count = 0;
for (i = 0; i < rows; ++i) {
       for (j = 0; j < cols - i; ++j) {
              Arr[i][j] = ++count;
              cout << Arr[i][j] << " ";
       }
       cout << endl;
}
cout << endl;
for (i = 0; i < rows; ++i)
       cout << *Arr[i] << " ";
cout << endl;
for (i = 0; i < rows; ++i)
       cout << *(Arr[i] + i) << " ";
cout << endl;
cout << Arr << " " << *Arr << " " << **Arr << endl;
cout << Arr[3] << " " << *(Arr + 3) << " " << *Arr[3] << endl;
cout << Arr[2] + 1 << endl;
cout << *Arr[3] + 2 << endl;
cout << *(*Arr + 2) + 1 << endl;
cout << *(Arr + 2)[3] << endl;
for (i = 0; i < rows; ++i)
       delete[] Arr[i];
delete[] Arr;
```

Answer:

As error occurs on the line"`cout << *(Arr + 2)[3] << endl;`" because the value that we are trying to access is out of the "Arr" array's range. It can be corrected by changing the line"`cout << *(Arr + 2)[3] << endl;`" with the line"`cout << *(*(Arr + 2) + 3) << endl;`".
Now the attained output will be:

```
1_2_3_4_5_6_7
8_9_10_11_12_13
14_15_16_17_18
19_20_21_22
```

```
1_8_14_19
1_9_16_22

Address of Arr[0]_Address of Arr[0][0]_1
Address of Arr[3][0]_Address of Arr[3][0]_19
Address stored in Arr[2][1]
21
4
17

where "_" is for the "cout << " ";" that is being printed.
```

Q-3

(1)

```
void reverse(char * s) {

	int size = 0;
	int i = 0;
	for (; s[i] != '\0'; i++) {
		size++;
	}

	i = 0;
	for (; i < size; i++) {
		for (int j = i + 1; j < size; j++) {
			while (s[i] == s[j]) {
				for (int x = j; x < size; x++) {
					s[x] = s[x + 1];
				}
				size--;
			}
		}
	}
	s[size] = '\0';

	for (i = 0; i < size / 2; i++) {
		char temp = s[i];
		s[i] = s[size - i - 1];
		s[size - i - 1] = temp;
	}
}
```

(2)

CODE:

```
int main() {
	char *a[] = { "Argentina", "Korea", "Greece", "Nigeria" };
	cout << *(a + 1) << endl;
	cout << *a[0] << endl;
	cout << a[3] << endl;
	cout << a[3][1] << endl;
	return 0;
}
```

Answer:

```
Korea
A
Nigeria
i
```

(3)

CODE:

```
int main() {
	const char* what = "Is This";
	what = "Interesting";
	cout << *what;
	what[3] = 'a';
	cout << *what;
}
```

Answer:

As error occurs on the line” `what[3] = 'a';`” because all the charactors of the array” `what`” are constants and can only be assigned only at time of declaration and cannot be reassigned.
So the expected output will be:

```
I
```

(4)

Answer:

In “`char *const A = "Hi";`”, the pointer “`A`” is a constant pointer assigned to the address of a character array, it can’t be assigned to address of another character array or character after being assigned once. Whereas in “`const char* B = "Hi";`”, the pointer “`B`” is a simple pointer assigned to a constant character array, the elements of that constant character array can only be assigned only at time of declaration and can’t be reassigned or can’t be modified after being assigned only.

(5)

Answer:

In “`const char* C = "hi mom"; C[3] = 'a';`”, the pointer “`C`” is a simple pointer assigned to a constant character array, the elements of that constant character array can only be assigned only at time of declaration and can’t be reassigned or can’t be modified after being assigned only, so when character’a’ is being assigned to `C[3]`, it gives Error. Whereas in “`char *const D = "hi mom"; D = "hi dad";`”, the pointer “`D`” is a constant pointer assigned to the address of a character array, it can’t be assigned to any other address of another array or character after being assigned once so when the address of the character array “`"hi dad"`” is being assigned to `D`, it gives Error.

(6)

CODE:

```
char ** s = new char *[1];
char * name = new char[20];
strcpy(name, "John Doe");
s[0] = name;
delete[] name;
cout << s[0] << endl;
delete[] s;
s = nullptr;
```

Answer:

After deleting "`name`", it becomes a dangling pointer as it has not been assigned the address of "`NULL`" after deletion. `s[0]` will cout a garbage value when it is cout after deleting the pointer "`name`" as the address of "name" has been deleted. No memory leak can be found.

THANK YOU 😊