Ministry of Higher Education
Colleges of Applied Sciences
IT Department

Course: SFDV4001 – Object-Oriented Programming & User Interface
Lab: 2 – Pointer and Array

1. Pointer

Observe the code listing 1.1, and answer the following questions.

- Should a pointer points to a value or an address of a variable ?
- Can the pointer be used to change the value of a variable ?

Code listing 1.1

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

int main()
{
     int a=1;
     int b=2;
     int c[3]={3,4,5};
     int *d = &a;
     int &e = a;

     a = b+c[0];
     b = a;
     c[1] = *d;
     *d = 7;
     e = 8;

     cout << "a=" << a << "\n";
}
```

2. Parameter Passing

Observe the code listing 2.1, and answer the following questions.

- What is the main difference between void value_parameter_func(int a) and void variable_parameter_func(int &a)?
- Is there any difference between variable_parameter_func(int &a) and void pointer_parameter_func(int *a)?

Code listing 2.1

```
#include <iostream>

using namespace std;
void value_parameter_func(int a) {
    a = 2;
    cout << "address = " << (unsigned int)&a << endl;
    cout << "value = " << a << endl ;
}
void variable_parameter_func(int &a) {
    a = 3;
    cout << "address = " << (unsigned int)&a << endl;
    cout << "value = " << a << endl ;
}
void pointer_parameter_func(int *a) {
    *a = 4;
    cout << "address = " << (unsigned int)a << endl;
    cout << "value = " << *a << endl ;
}
int main(int argc, char *argv[]) {
    int b=1;
    cout << "before value_parameter_func, b = " << b << endl;
    value_parameter_func(b);
    cout << "after value_parameter_func, b = " << b << endl << endl;
    cout << "before variable_parameter_func, b = " << b << endl;
    variable_parameter_func(b);
    cout << "after variable_parameter_func, b = " << b << endl << endl;
    cout << "before pointer_parameter_func, b = " << b << endl;
    pointer_parameter_func(&b);
    cout << "after pointer_parameter_func, b = " << b << endl << endl;
}
```

3. Dynamic Memory Allocation and Deallocation

Observe the code listing 3.1 and 3.2, and identify the problem. Then, find a solution to this problem.

Code listing 3.1

```
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
    int *n;
    for (int i=0; i<5e7; i++)
        n = new int;
    delete n;
}
```

Code listing 3.2

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

int main(int argc, char *argv[])
{
    int *n;
    n = new int;
    delete n;
    delete n;
}
```

4. Array

Observe the code listing 4.1, and learn how to create an array of integers, and initialize this array.

Code listing 4.1

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

int main()
{
    //Create an array of 10 elements
    int arr[10];
    cout << "values of arr elements before initialization " << endl;
    for (int i = 0; i < 10; i++)
        cout << arr[i] << endl;

    //Initialize the array elements to zeros
    for (int i = 0; i < 10; i++)
        arr[i] = 0;

    //Show the values after initialization
    cout << "values of arr elements after initialization " << endl;
    for (int i = 0; i < 10; i++)
        cout << arr[i] << endl;
}
```

5. Programming Exercise

Observe the code listing 5.1, and complete the missing code, so that the program can output even numbers between 0 and 10, i.e. [0,2,4,6,8,10]. In addition, the program should also count the even numbers found.

Hint: Use modulo operator **%** to find the even numbers.

Code listing 5.1

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

int main() {
    for (int i = 0; i <= 10 ; i++)
        // MISSING CODE
    return 0;
}
```