Karan Sahu
NetID: KXS190007
Partner:
Ogheneyoma Akoni
NetID: OXA180001
CS4337.503

Assignment 6

1-Present one argument against providing both static and dynamic local variables in subprograms.Write a report (at least 4 lines)

1. In subprograms local variables can be static or dynamic. One of the disadvantaged with static local variables is their inability to support recursion which happens to be the greatest disadvantage with static local variables. The main disadvantage with dynamic local variables is there is the cost of time to allocate, initialize, deallocate these variables for each call to the subprogram.

2-Consider the following program written in C syntax:

```
void fun (int first, int second)
{ first += first;
 second += second;
    }
void main() {
int list[2] = {1, 3};
fun(list[0], list[1]);
}
```

For each of the following parameter-passing methods, what are the values of the list array after execution?

- Passed by value
- Passed by reference
- Passed by value-result

2.

a. Passed by value: list[0] = 1 and list[1] = 3
b. Passed by reference: list[0] = 2 and list[1] = 6
c. Passed by value-result: list[0] = 2 and list[1] = 6

3-Write an abstract data type for queues whose elements store 10-character names. The queue elements must be dynamically allocated from the heap. Queue operations are enqueue, dequeue, and empty. Use either C++, Java, or C#.

Source code in separate folder

4-The following are design issues for subprograms. Explain the design choices in python for these issues. Please also provide enough examples to illustrate the design choices.

- Are local variables statically or dynamically allocated?
- Can subprogram definitions appear in other subprogram definitions?
- What parameter-passing method or methods are used?
- Are the types of the actual parameters checked against the types of the formal parameters?
- If subprograms can be passed as parameters and subprograms can be nested, what is the referencing environment of a passed subprogram?
- Can subprograms be overloaded?
- Can subprograms be generic?
- If the language allows nested subprograms, are closures supported?

1. Dynamically allocated- everything in Python is an object. This means that Dynamic Memory Allocation underlies Python Memory Management. When objects are no longer needed, the Python Memory Manager will automatically reclaim memory from them.

```
x = 1
def SubProgram1():
    x = 4
    print("local x = {}".format(x))
SubProgram1()
print("global x = {}".format(x))
```

```
local x = 4
global x = 1
```

2. Yes. A subprogram definition is a description of the actions of the subprogram abstraction. Since java can have nested subprograms this is possible.

```
def main():
    def Print5():
        a = 5
        print(a)
        def Print7():
            a = 7
            print(a)
        Print7()
    Print5()
main()
```

```
5
7
```

   Since subprogram Print 7 is within subprogram Print5 it carries its subprogram definition within it.
3. The parameter-passing method of python is called pass-by-assignment. Because all data values are object, ever variable is a reference to an object. Pass-by-assignment In python is in effect pass-by-reference. This does work for immutable types which will be shown in the given example

```
 6
 7  def main(x, arr):
 8      x=10
 9      arr[0] = 1
10  x=5
11  arr = [0, 0, 0]
12  print(x)
13  print(arr)
14  main(x, arr)
15  print(x)
16  print(arr)
```

```
TERMINAL   PROBLEMS   OUTPUT

Microsoft Windows [Version 10.
(c) 2019 Microsoft Corporation

C:\Users\karan>python -u "c:\U
5
[0, 0, 0]
5
[1, 0, 0]
```

4. No there is no parameter checking because of pythons nature. you dont specify the type in the actual or formal parameter. (teacher said no example if its not implemented)

5. Yes python can pass subprograms as parameters and you can have nested sub programs. It is the environment of the call statement that enacts the passed subprogram "Shallow binding."

```
var = 20
def my_method(v):
    v += 10
    return v
def add10(x):
    x = x + 10
    return x
print(add10(my_method(var)))
```

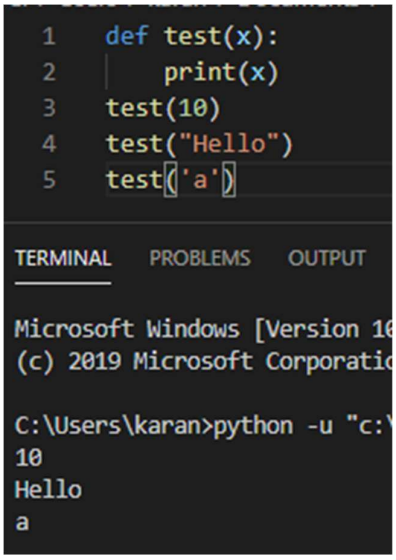
```
40
```

6. No a program Cannot be overloaded as demonstrated below.

```
1   def PrintString(x, y, z):
2       print(x)
3       print(y)
4       print(z)
5
6   def PrintString(x, y):
7       print(x)
8       print(y)
9
10  def PrintString(x):
11      print(x)
12
13  PrintString("Hello")
14  PrintString("Whats", "Up")
15  PrintString("Nice", "To", "Meet")
16
17
```

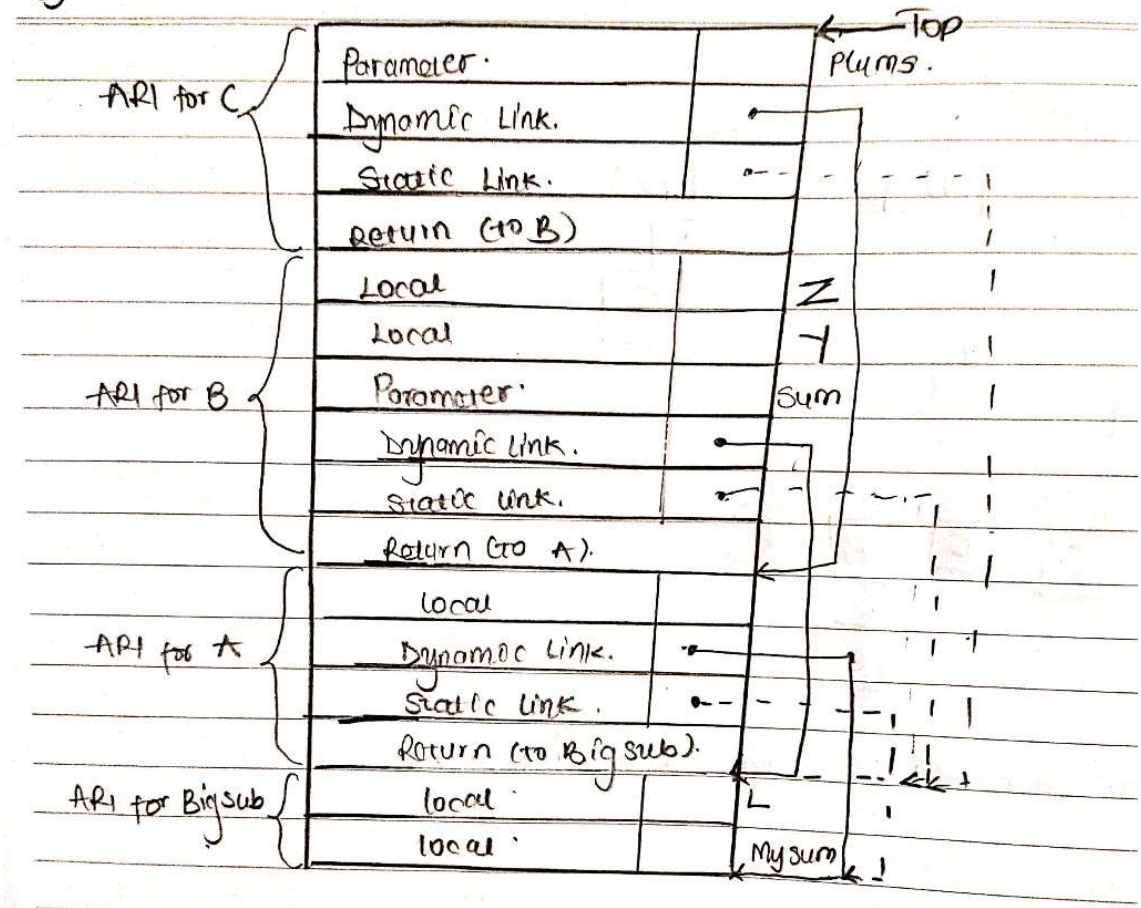
```
ERMINAL   PROBLEMS   OUTPUT   DEBUG CONSOLE
icrosoft Windows [Version 10.0.18363.1198]
c) 2019 Microsoft Corporation. All rights reserved.

:\Users\karan>python -u "c:\Users\karan\Documents\UTD Semesters\UTD fall 2020\CS 4
ello
raceback (most recent call last):
 File "c:\Users\karan\Documents\UTD Semesters\UTD fall 2020\CS 4337\Assignment\Assi
   PrintString("Whats", "Up")
ypeError: PrintString() takes 1 positional argument but 2 were given
```

7. Yes. A generic subprogram takes parameters of differenrnt types on different activations. Python does not make you specify the type for the paramter allowing you to do this as I will show in the example below.

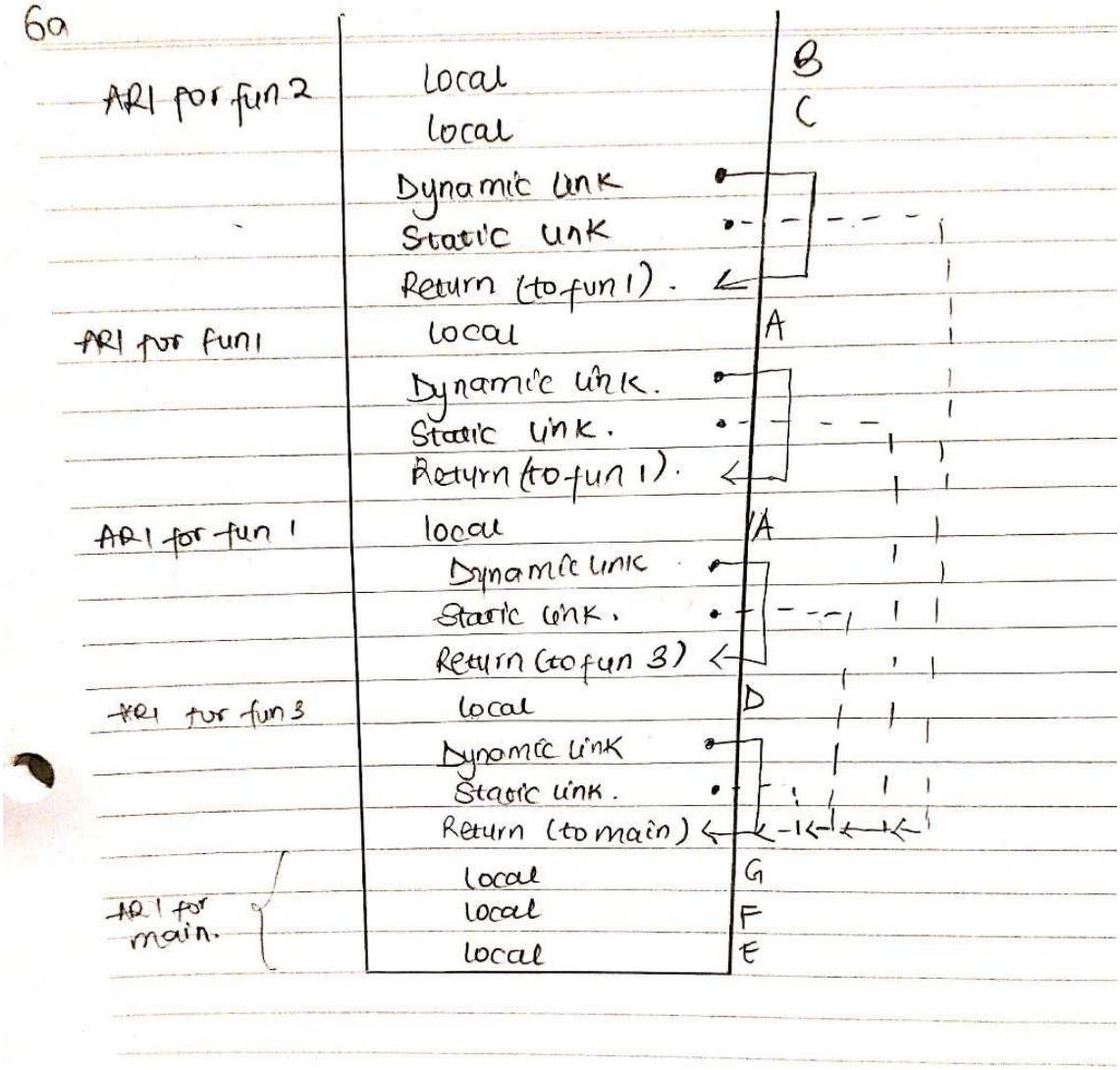


8.Yes. A Closure is a function object that remembers values in enclosing scopes even if they are not present in memory. You can see this demonstrated in python in the following code. Nonlocal variables are read only.

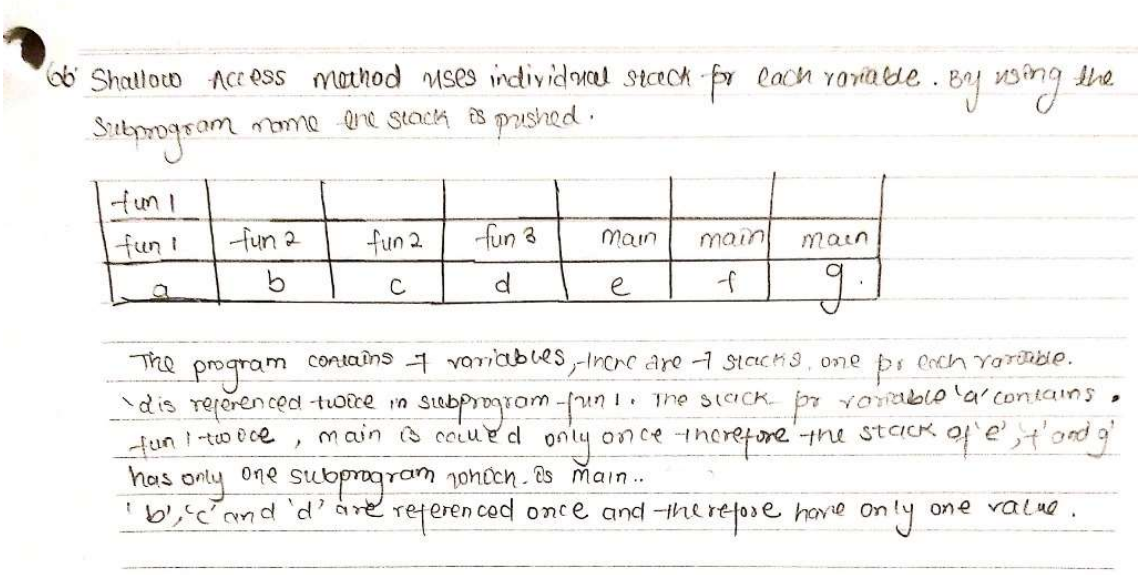
5)


5)
Top
Plums.
AR1 for C
Parameter.
Dynamic Link.
Static Link.
Return (to B)
AR1 for B
Local
Local
Parameter.
Dynamic Link.
Static Link.
Return (to A).
AR1 for A
Local
Dynamic Link.
Static Link.
Return (to Bigsub).
AR1 for Bigsub
local.
local.
Z
Y
Sum
L
Mysum

6a)


6a
AR1 for fun2
local
local
Dynamic link
Static link
Return (to fun1).
B
C
AR1 for fun1
local
Dynamic link.
Static link.
Return (to fun1).
A
AR1 for fun1
local
Dynamic link
Static link.
Return (to fun3)
A
AR1 for fun3
local
Dynamic link
Static link.
Return (to main)
D
AR1 for main.
local
local
local
G
F
E

6b)



6b) Shallow Access method uses individual stack for each variable. By using the subprogram name the stack is pushed.

fun 1						
fun 1	fun 2	fun 2	fun 3	main	main	main
a	b	c	d	e	f	g.

The program contains 7 variables, hence are 7 stacks, one for each variable.
'a' is referenced twice in subprogram fun 1. The stack for variable 'a' contains fun 1 twice, main is called only once therefore the stack of 'e', 'f' and 'g' has only one subprogram which is main..
'b', 'c' and 'd' are referenced once and therefore have only one value.

7)

a)Primitive types and Objects:
A primitive type is a data type where the values that it can represent have a very simple nature (a number, a character or a truth-value); the primitive types are the most basic building blocks for any programming language and are the base for more complex data types. The primitive types available in C++:

- Integer: Keyword used for integer data types is int. Integers typically requires 4 bytes of memory space and ranges from -2147483648 to 2147483647.
- Character: Character data type is used for storing characters. Keyword used for character data type is char. Characters typically requires 1 byte of memory space and ranges from - 128 to 127 or 0 to 255.
- Boolean: Boolean data type is used for storing Boolean or logical values. A Boolean variable can store either *true* or *false*. Keyword used for Boolean data type is bool.
- Floating Point: Floating Point data type is used for storing single precision floating point values or decimal values. Keyword used for floating point data type is float. Float variables typically requires 4 byte of memory space.
- Double Floating Point: Double Floating-Point data type is used for storing double precision floating point values or decimal values. Keyword used for double floating-point data type is double. Double variables typically requires 8 byte of memory space.
- void: Void means without any value. void datatype represents a valueless entity. Void data type is used for those function which does not returns a value.
- Wide Character: Wide character data type is also a character data type, but this data type has size greater than the normal 8-bit datatype. Represented by wchar_t. It is generally 2 or 4 bytes long.

An object is an instance of a class. An object in OOPS is nothing but a self-contained component which consists of methods and properties to make a particular type of data useful.

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

int main()
{
    cout << "char size: "<< sizeof(char) <<" byte"<<endl;
    cout << "int size: "<< sizeof(int) <<" bytes"<<endl;
    cout << "short int size: "<< sizeof(short int) <<" bytes"<<endl;
    cout << "long int size: "<< sizeof(long int) <<" bytes"<<endl;
    cout << "float size: "<< sizeof(float) <<" bytes"<<endl;
    cout << "double size: "<< sizeof(double) <<" bytes"<<endl;
    cout << "wide char size: "<< sizeof(wchar_t) <<" bytes"<<endl;
    cout << "signed long int size: "<< sizeof(signed long int) <<" bytes"<<endl;
    cout << "unsigned long int size: "<< sizeof(unsigned long int)<< " bytes"<<endl;
   return 0;
}
```

```
char size: 1 byte
int size: 4 bytes
short int size: 2 bytes
long int size: 8 bytes
float size: 4 bytes
double size: 8 bytes
wide char size: 4 bytes
signed long int size: 8 bytes
unsigned long int size: 8 bytes
```

b)Are Subclasses Subtypes:

There are important differences between subtypes and subclasses in supporting reuse. Subclasses allow one to reuse the code inside classes - both instance variable declarations and method definitions. Thus, they are useful in supporting code reuse *inside* a class. Subtyping on the other hand is useful in supporting reuse externally, giving rise to a form of polymorphism. That is, once a data type is determined to be a subtype of another, any function or procedure that could be applied to elements of the supertype can also be applied to elements of the subtype.

Example:
Class courses: public Major – courses is the name of a class which is a subclass of a derived class of the Major class.
Both classes are having polymorphic functions which have the same name called “print” but the definitions of the functions are different.
When print is executed from x, the object of major class, it will print a statement about the major while from y, the object of courses class, it will print a statement about the course.
Major-base or parent class.
Courses – derived or child class

```cpp
#include <iostream>
using namespace std;
class Major
{
    string name;
    public:
        string getname()
        {
            return name;
        }
        void setname(string y)
        {
            name = y;
        }
        void print()
        {
            cout << "my major is SE"<< endl;
        }
};
class courses: public Major
{
    public:
        void print()
        {
            cout << "I offer CS 4337"<< endl;
        }
};

int main()
{
   Major x;
   courses y;
   x.print();
   y.print();
   return 0;
}
```

```
my major is SE
I offer CS 4337
```

c)Single and Multiple Inheritance:

Single inheritance is one in which the derived class inherits the single base class either publicly, privately or protectedly. In single inheritance, the derived class uses the features or members of the single base class. These base class members can be accessed by derived class or child class according to the access specifier specified during inheriting the parent class or base class. In C++, it is possible to inherit attributes and methods from one class to another. Inheritance concept is grouped into two categories: derived class (child) - the class that inherits from another class base class (parent) - the class being inherited from

Multiple inheritance is one in which the derived class acquires two or more base classes. In multiple inheritance, the derived class are allowed to use the joint features of the inherited base classes.

Example of Single Inheritance:

```
#include <iostream>

using namespace std;
class furniture{
    public:
        string type = " chair";
        void material()
        {
            cout << "wood \n";
        }
};

class Table: public furniture
{
    public:
        string use = "centre table";
};

int main()
{
   Table obj;
   obj.material();
   cout << obj.type + " "+  obj.use;
   return 0;
}
```

```
wood
 chair centre table
```

Example of Multiple inheritance:

```cpp
#include <iostream>
using namespace std;
class first{
    public:
        void f1()
        {
            cout<<"parent class here \n";
        }
};
class second{
    public:
            void f2()
            {
                cout << "another class here \n";
            }
};
class Child: public first, public second
{
};
int main()
{
   Child obj;
   obj.f1();
   obj.f2();
   return 0;
}
```

```
parent class here
another class here
```

d)Allocation and Deallocation of objects in C++:

In addition to Static variables and Automatic variables, C and C++ provide a third category of variables known as Dynamic variables**.** Any global variable is static, as is any local variable explicitly declared as static. The lifetime of a static variable is the lifetime of the program

Memory allocation is accomplished using the new operator and deallocation is accomplished using the delete operator. Normally, the new operator creates an uninitialized variable of the specified type and returns a pointer to it. If, however, there is insufficient memory available, the new operator returns a NULL pointer. Variables created dynamically are said to be on the *free store* or *heap*. A dynamic variable is unnamed and cannot be directly accessed. It must be indirectly accessed through the pointer returned by new. Dynamic variables can be destroyed at any time during program execution. The delete operator is used for this purpose.

Example:

```
#include <iostream>

using namespace std;


int main()
{
    int *a = NULL;
    a = new(nothrow) int;
    if(!a)
        cout<< "memory allocation failed \n";
    else
    {
        *a= 5;
        cout << "a = "<< *a<< endl;
    }

    float *b = new float(10.5);
    cout << " b = " << *b << endl;

    int e=4;
    int *d = new(nothrow) int [e];
    if(!d)
        cout<< "memory allocation failed \n";
```

```
    else
    {
        for(int i =0; i < e; i++)
        {
            d[i] = i+1;
        }
        cout << "storage of value in block memory: ";
        for(int i = 0; i < e; i++)
        {
            cout << d[i]<< " ";
        }
    }

    delete a;
    delete b;
    delete[] d;
    return 0;
}

```

```
a = 5
 b = 10.5
storage of value in block memory: 1 2 3 4
```

e)Dynamic and Static Binding:

Static Binding happens at the compile-time and Dynamic Binding happens at the runtime. In static binding, the function definition and the function call are linked during the compile-time whereas in dynamic binding the function calls are not resolved until runtime. So they are not bound until runtime.

Example that illustrates Dynamic Binding:

```cpp
#include <iostream>

using namespace std;

class first
{
    public:

    virtual void a()
    {
        cout << "Base class function \n";
    }
};

class B: public first
{
    public:

    void a()
    {
        cout << "Derived class function called \n";
    }
};
```

Example to illustrate Static Binding:

```cpp
#include <iostream>

using namespace std;

class first
{
    public:

    int sum (int x, int y)
    {
        return x+y;
    }
    int sum(int x, int y, int z)
    {
        return x+y+z;
    }
};

int main()
{
    first obj;
    cout << "first sum = "<< obj.sum(2,3)<<'\n';
    cout << "second sum = "<< obj.sum(2,3,5)<<'\n';
    return 0;
}

```

```
first sum = 5
second sum = 10
```

```
int main()
{
    first base;
    B derived;

    first *bptr = &base;
    bptr->a();

    bptr = &derived;
    bptr->a();

    return 0;
}
```

```
Base class function
Derived class function called
```

f)Nested Classes:
A nested class is a class which is declared in another enclosing class. In a nested class is a member and has the same access rights as any other member. Members of an enclosing class have no special access to members of a nested class.

Example:

Program executes successfully.

```cpp
#include <iostream>

using namespace std;

class Nested{
    private:int a;
    class Nested2{
        int b;
        void Nested3(Nested *c)
        {
            cout << c->a;
        }
    };
};

int main()
{

}
```

```
...Program finished with exit code 0
Press ENTER to exit console.
```

g)Initialization:

A constructor is a member function of a class which initializes objects of a class. In C++, Constructor is automatically called when object(instance of class) create. constructor name is same as class name. It help in initialise the value or message just starting the program.

```cpp
#include <iostream>
using namespace std;
class MyClass {     // The class
  public:           // Access specifier
    MyClass() {     // Constructor
      cout << "Hello World!";
    }
};

int main() {
  MyClass myObj;
  return 0;
}
```

```
Hello World!
```