Why C++?

- Popular and relevant (used in nearly every application domain):
  - end-user applications (Word, Excel, PowerPoint, Photoshop, Acrobat, Quicken, games)
  - operating systems (Windows 9x, NT, XP; IBM's K42; some Apple OS X)
  - large-scale web servers/apps (Amazon, Google)
  - central database control (Israel's census bureau; Amadeus; Morgan-Stanley financial modeling)
  - communications (Alcatel; Nokia; 800 telephone numbers; major transmission nodes in Germany and France)
  - numerical computation / graphics (Maya)
  - device drivers under real-time constraints
- Stable, compatible, scalable

C vs. C++ - 1

- C is a system programming language, whereas C++ is a general-purpose programming language commonly used in embedded systems. C is procedural, so it doesn't support classes and objects like C++ does (although, despite being object-oriented, C++ can be procedural like C, making it a bit more hybrid).
- Generally, you'd opt to use C over C++ if you didn't want the extra overhead of C++—however you can always just pick the features of C++ you want to use and exclude the others.

C vs. C++ - 2

- C++ is a superset of C
- C++ has all the characteristics of C
- Object-oriented! (Encapsulation, Data hiding, Inheritance, Polymorphism)
- Encapsulation: "black box" – internal data hidden. **Inheritance:** related classes share implementation and/or interface. **Polymorphism:** ability to use a class without knowing its type
- C++ is C incremented
  (orig., "C with classes")
- C++ is more *expressive*
  (fewer C++ source lines needed than C source lines for same program)
- C++ is just as *permissive*
  (anything you can do in C can also be done in C++)
- C++ can be just as *efficient*
  (most C++ expressions need no run-time support;
  C++ allows you to
  - manipulate bits directly and interface with hardware without regard for safety or ease of comprehension, BUT
  - hide these details behind a safe, clean, elegant interface)
- C++ is more *maintainable*
  (*1000 lines of code* – even brute force, spaghetti code will work;
  100,000 lines of code – need good structure, or new errors will be introduced as quickly as old errors are removed)

Design goals of C++

- Backward compatibility with C
  (almost completely – every program in K&R is a C++ program – but additional keywords can cause problems)
- Simplicity, elegance
  (few built-in data types, e.g., no matrices)
- Support for user-defined data types
  (act like built-in types; N.B. Standard Template Library (STL))
- No compromise in efficiency, run-time or memory
  (unless "advanced features" used)
- Compilation analysis to prevent accidental corruption of data
  (type-checking and data hiding)
- Support object-oriented style of programming

C++ Basic Syntax

When we consider a C++ program, it can be defined as a collection of objects that communicate via invoking each other's methods. Let us now briefly look into what a class, object, methods, and instant variables mean.

- **Object** – Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behaviors - wagging, barking, eating. An object is an instance of a class.
- **Class** – A class can be defined as a template/blueprint that describes the behaviors/states that object of its type support.
- **Methods** – A method is basically a behavior. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.
- **Instance Variables** – Each object has its unique set of instance variables. An object's state is created by the values assigned to these instance variables.

C++ Program Structure-1

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

// main() is where program execution begins.
int main() {
   cout << "Hello World"; // prints Hello World
   return 0;
}
```


C++ Program Structure-2

- The C++ language defines several headers, which contain information that is either necessary or useful to your program. For this program, the header **<iostream>** is needed.
- The line **using namespace std;** tells the compiler to use the std namespace. Namespaces are a relatively recent addition to C++.
- The next line '**// main() is where program execution begins.**' is a single-line comment available in C++. Single-line comments begin with // and stop at the end of the line.
- The line **int main()** is the main function where program execution begins.
- The next line **cout << "This is my first C++ program.";** causes the message "This is my first C++ program" to be displayed on the screen.
- The next line **return 0;** terminates main( )function and causes it to return the value 0 to the calling process.

Compile and Execute C++ Program (Linux)

Let's look at how to save the file, compile and run the program. Please follow the steps given below –

- Open a text editor and add the code as above.
- Save the file as: hello.cpp
- Open a command prompt and go to the directory where you saved the file.
- Type 'g++ hello.cpp' and press enter to compile your code. If there are no errors in your code the command prompt will take you to the next line and would generate a.out executable file.
- Now, type 'a.out' to run your program.
- You will be able to see ' Hello World ' printed on the window.

Compile and Execute C++ Program (Linux)

```
$ g++ hello.cpp
$ ./a.out
Hello World
```

- Make sure that g++ is in your path and that you are running it in the directory containing file hello.cpp.
- You can compile C/C++ programs using makefile.

C++ Identifiers

- A C++ identifier is a name used to identify a variable, function, class, module, or any other user-defined item. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores, and digits (0 to 9).
- C++ does not allow punctuation characters such as @, $, and % within identifiers. C++ is a case-sensitive programming language. Thus, **Manpower** and **manpower** are two different identifiers in C++.
- Here are some examples of acceptable identifiers –

mohd zara abc move_name a_123
myname50 _temp j a23b9 retVal

Initializing Local and Global Variables

When a local variable is defined, it is not initialized by the system, you must initialize it yourself. Global variables are initialized automatically by the system when you define them as follows –

Data Type	Initializer
int	0
char	'\0'
float	0
double	0
pointer	NULL

Storage Classes in C++

A storage class defines the scope (visibility) and life-time of variables and/or functions within a C++ Program. These specifiers precede the type that they modify. There are following storage classes, which can be used in a C++ Program

- auto
- register
- static
- extern
- mutable

The auto Storage Class

The **auto** storage class is the default storage class for all local variables.

```
{
   int mount;
   auto int month;
}
```

The example above defines two variables with the same storage class, auto can only be used within functions, i.e., local variables.

The register Storage Class

The **register** storage class is used to define local variables that should be stored in a register instead of RAM. This means that the variable has a maximum size equal to the register size (usually one word) and can't have the unary '&' operator applied to it (as it does not have a memory location).

```
{
   register int  miles;
}
```

The register should only be used for variables that require quick access such as counters. It should also be noted that defining 'register' does not mean that the variable will be stored in a register. It means that it MIGHT be stored in a register depending on hardware and implementation restrictions.

The static Storage Class-1

- The **static** storage class instructs the compiler to keep a local variable in existence during the life-time of the program instead of creating and destroying it each time it comes into and goes out of scope. Therefore, making local variables static allows them to maintain their values between function calls.
- The static modifier may also be applied to global variables. When this is done, it causes that variable's scope to be restricted to the file in which it is declared.
- In C++, when static is used on a class data member, it causes only one copy of that member to be shared by all objects of its class.

The static Storage Class-2

```
#include <iostream>
void func(void);
static int count = 10; /* Global variable */
main() {
   while(count--) {
      func();
   }
   return 0;
}
void func( void ) {
   static int i = 5; // local static variable
   i++;
   std::cout << "i is " << i ;
   std::cout << " and count is " << count << std::endl;
}
```

```
i is 6 and count is 9
i is 7 and count is 8
i is 8 and count is 7
i is 9 and count is 6
i is 10 and count is 5
i is 11 and count is 4
i is 12 and count is 3
i is 13 and count is 2
i is 14 and count is 1
i is 15 and count is 0
```

The extern Storage Class-1

- The **extern** storage class is used to give a reference of a global variable that is visible to ALL the program files. When you use 'extern' the variable cannot be initialized as all it does is point the variable name at a storage location that has been previously defined.
- When you have multiple files and you define a global variable or function, which will be used in other files also, then *extern* will be used in another file to give reference of defined variable or function. Just for understanding *extern* is used to declare a global variable or function in another file.
- The extern modifier is most commonly used when there are two or more files sharing the same global variables or functions as explained below.

The extern Storage Class-2

First File: main.cpp

```
#include <iostream>
int count ;
extern void write_extern();
main() {
   count = 5;
   write_extern();
}
```

Second File: support.cpp

```
#include <iostream>
extern int count;
void write_extern(void) {
   std::cout << "Count is " <<
count << std::endl;
}
```

Here, extern keyword is being used to declare count in another file. Now compile these two files as follows –
$g++ main.cpp support.cpp -o write
This will produce write executable program, try to execute write and check the result as follows –
$./write
5