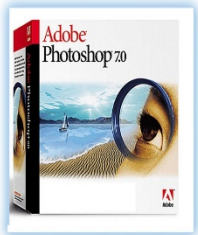
Introduction to C++

Jinsoo Jang

Programming Languages

C Language

- Created by Dennis Ritchie, AT&T Bell Labs in 1970s
- International standard ISO/IEC 9899:2018 (informally known as “C18”)
- Available on wide range of platforms, from microcontrollers to supercomputers; very few platforms for which C compiler not available
- Procedural, provides language constructs that map efficiently to machine instructions
- Does not directly support object-oriented or generic programming
- Application domains: system software, device drivers, embedded applications, application software
- Greatly influenced development of C++



Credit: Google books – Lecture slides for programming in c++

C++

- Created by Bjarne Stroustrup, Bell Labs
- Originally C with Classes, renamed as C++ in 1983
- Most recent specification of language in ISO/IEC 14882:2017 (informally known as “C++17”)
- Loosely speaking is superset of C
- Directly supports object-oriented and generic programming
- Application domains: systems software, application software, device drivers, embedded software, high-performance server and client applications, entertainment software such as video games, native code for Android applications
- Greatly influenced development of C# and Java

Credit: Google books – Lecture slides for programming in c++

Java

- Developed in 1990s by James Gosling at Sun Microsystems (later bought by Oracle Corporation)
- de facto standard but not international standard
- Usually less efficient than C and C++
- Simplified memory management (with garbage collection)
- Direct support for object-oriented programming
- Application domains: web applications, Android applications

Credit: Google books – Lecture slides for programming in c++

Fortran

- Designed by John Backus, IBM, in 1950s
- International standard ISO/IEC 1539-1:2010 (informally known as ”Fortran 2008”)
- Application domain: scientific and engineering applications
  - Supercomputing tasks such as weather and climate modelling, finite element analysis, computational fluid dynamics, computational physics, computational chemistry

First mass-produced

Used for programming an IBM 704 mainframe computer

Credit: Google books – Lecture slides for programming in c++

C#

- Developed by Microsoft, team led by Anders Hejlsberg
- ECMA-334 and ISO/IEC 23270:2006
- Standardized by ECMA or ISO/IEC
- Intellectual property concerns over Microsoft patents
- Object oriented

Windows 10

Media player

Beagle
Search system on Linux

Credit: Google books – Lecture slides for programming in c++

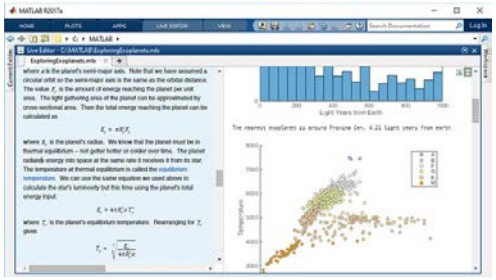
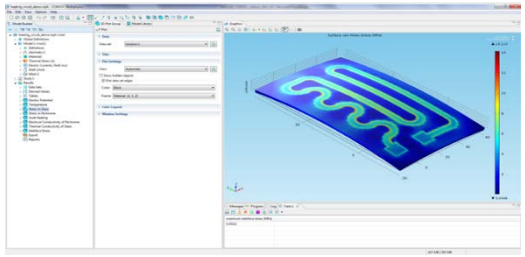
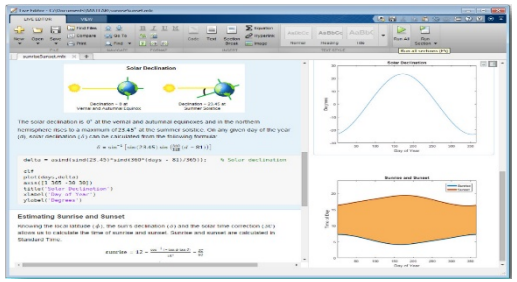
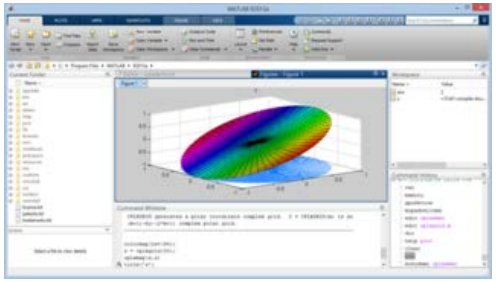
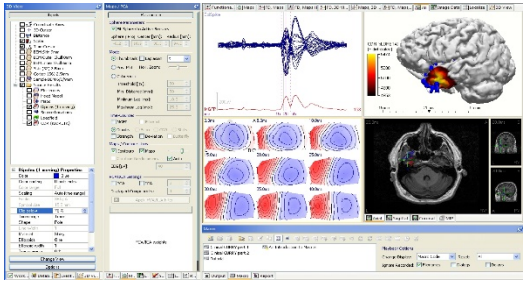
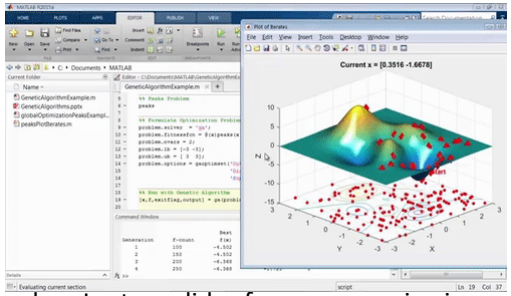
Objective C

- Developed by Tom Love and Brad Cox of Stepstone (later bought by NeXT and subsequently Apple)
- Used primarily on Apple Mac OS X and iOS
- Strict superset of C
- No official standard that describes Objective C
- Authoritative manual on Objective-C 2.0 available from Apple

Credit: Google books – Lecture slides for programming in c++

MATLAB

- Proprietary language, developed by The MathWorks
- Not general-purpose programming language
- Application domain: numerical computing
- Used to design and simulate systems
- Not used to implement real-world systems

Credit: Google books – Lecture slides for programming in c++

Advantage of learning C++

- Vendor neutral : same in any platform or compiler
- International standard
- General purpose
- Powerful yet efficient

From Wikipedia

- Loosely speaking, includes C as subset; so can learn two languages (C++ and C) for price of one
- Easy to move from C++ to other languages but often not in other direction
- Consistently ranks amongst top languages in TIOBE Software Programming Community Index

Credit: Google books – Lecture slides for programming in c++

Advantage of learning C++

❖ TIOBE Index for August 2019

Aug 2019	Aug 2018	Change	Programming Language	Ratings
1	1		Java	16.028%
2	2		C	15.154%
3	4	↑	Python	10.020%
4	3	↓	C++	6.057%
5	6	↑	C#	3.842%
6	5	↓	Visual Basic .NET	3.695%
7	8	↑	JavaScript	2.258%
8	7	↓	PHP	2.075%
9	14	⇈	Objective-C	1.690%
10	9	↓	SQL	1.625%
11	15	⇈	Ruby	1.316%
12	13	↑	MATLAB	1.274%

Disadvantage of C++

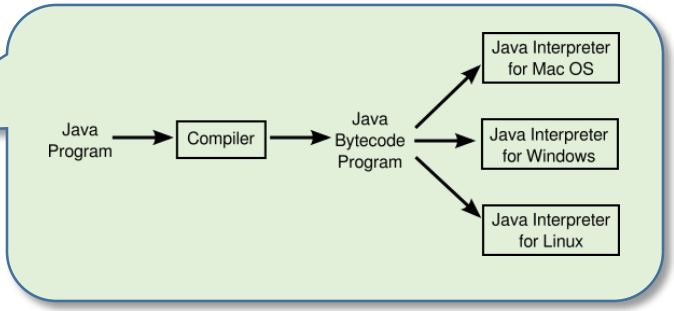
- Security problem
  - e.g., using pointers, friend function, and global variable
- Complex syntax
- No garbage collection

- Web applications in C++ are complex and difficult to debug

Java vs. C++

C++	VS	JAVA
C++ is platform dependent		Java is platform independent
It uses Compiler only		It uses Compiler and Interpreter both
It supports operator overloading		It does not supports operator overloading
C++ supports multiple inheritance.		Java dose not support multiple inheritance.
C++ supports Header Files.		It does not support header files. It uses the import keyword to include classes.

visit : www.studentlearningpoint.com



Language to Learn

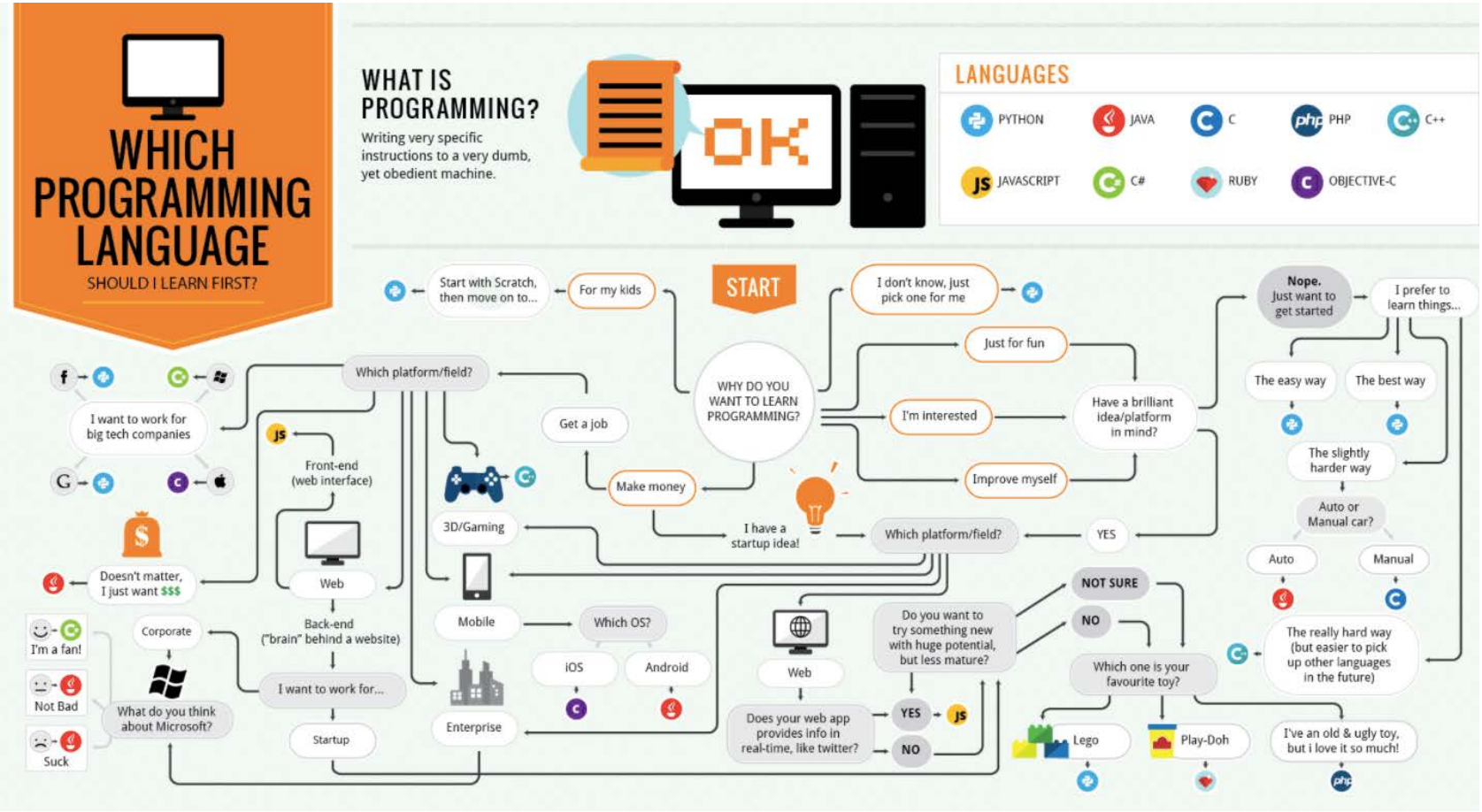
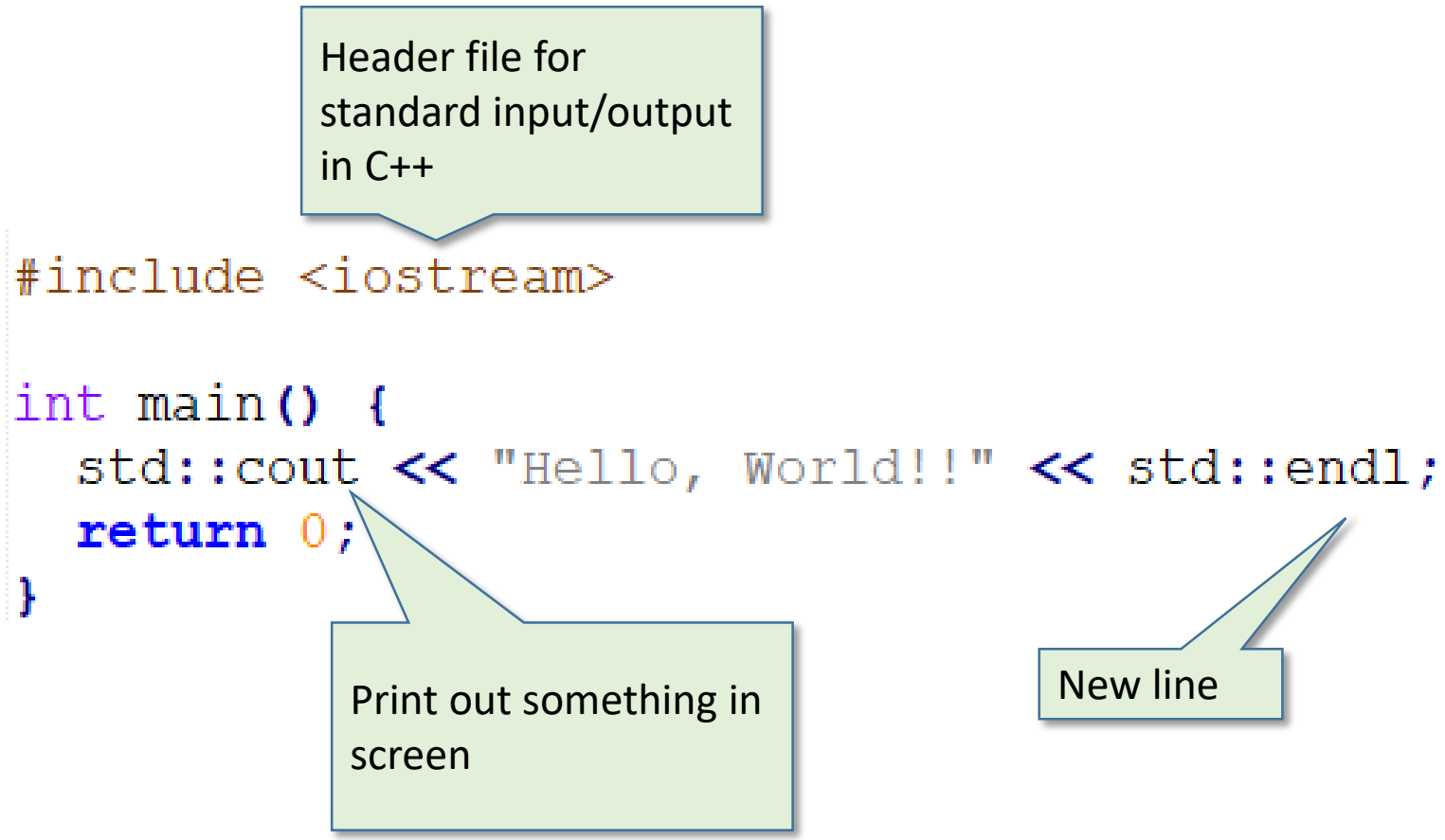


Figure from: https://codeburst.io/what-programming-language-should-i-learn-f3f164ca376c

Hello World

Hello, World


Header file for standard input/output in C++
#include <iostream>
int main() {
std::cout << "Hello, World!!" << std::endl;
return 0;
}
Print out something in screen
New line

Namespace

- Namespace is a declarative region that provides a scope to the identifiers (the name of types, functions, variables, etc.)
  - In a same namespace: identifiers are visible to one another without qualification
  - In the different namespace: require fully qualified name for each identifier

```
#include <iostream>

int main() {
  std::cout << "Hello, World!!" << std::endl;
  return 0;
}
```

Namespace “std”

Namespace cont'd

- Rules to create namespace
  - *Namespace* definition must be done at global scope, or nested inside another *Namespace*
  - Namespace definition doesn't terminates with a semicolon just like class definition
  - Namespace with Alias name

```
namespace cnucomputerscience {
    void program();
}
// cnucs is now alias for computerscience
namespace cnucs = cnucomputerscience;
```

  - Instance of Namespace is not creatable

Namespace cont'd

- Rules to create namespace (cont'd)
  - Unnamed namespace is supported

```
namespace  {
    void program();
    class functionA {};
}
```

Like using "static" keyword, program() and functionA are only accessible in this file

  - Namespace definition spans over multiple files, they are not redefined or overriden

```
//header1.h
namespace code
{
    void program();
    class tutorial{    };
    int x, y;
}

//header2
#include "header1.h";
namespace code
{
    void store();
    class library{    };
    int a,b;
}
```

Namespace cont'd

❖ Global namespace

```
#include <iostream>

int a = 100;

int main() {
    int a = 200;

    std::cout << "local a is: " << a << std::endl;
    std::cout << "global a is: " << ::a << std::endl;

    system("pause");
    return 0;
}
```

This a is in the global scope;

Look at how the global namespace is accessed

Namespace cont'd

❖ Global namespace cont'd

```
#include <iostream>

int a = 10;

namespace N
{
    int a = 100;

    void f()
    {
        int a = 1000;
        std::cout << a << std::endl;      //prints 1000
        std::cout << N::a << std::endl;   //prints 100
        std::cout << ::a << std::endl;    //prints 10
    }
}

int main() {
    N::f();
    system("pause");
}
```

- Scope denotes the lifetime of an object, for example,
  - A global variable exists as long as your program executes
  - A variable with a block scope exists as long as that block of code executes

Code from: stackoverlfow.com

C and C++ in common

C Style Support in C++

❖ Support C style declaration

```
#include <iostream>

int main() {
  int i;
  char c;
  double d;
  float f;

  return 0;
}
```

C Style Support in C++

- Pointers

```
int arr[10];
int *p1 = arr;

int i;
int *pi = &i;
```

- Pointer and array

```
int a[10];
int *p;
p=a; //legal
a=p; //illegal
```

```
int a[10];
int *p;
p++; //legal
a++; //illegal
```

C Style Support in C++

❖ Call by value

```
/*Call by value: InfoBrother*/

#include <iostream>
using namespace std;

//function, used to change the value of actual variable:
void increment(int x)
{
    ++x; //increment the value.
}

main()
{
    int x=10; //actual variable.
    cout<<" Before increment: "<<x<<endl;

    increment( x ); //function calling. call by value:
    cout<<" After increment: "<<x<<endl;

    return 0;
}
```

C Style Support in C++

❖ Call by pointer

```
 /*call by Pointer: InfoBrother*/

#include <iostream>
using namespace std;

//function, used to change the value of actual variable:
void increment(int *x)
{
    ++*x; //increment the value.
}

main()
{
    int x=10; //actual variable.
    int *ptr = &x;
    cout<<" Before increment: "<<x<<endl;

    increment( &x); //function calling. call by pointer:(pass address)
    cout<<" After increment: "<<x<<endl;

        return 0;
}
```

C Style Support in C++

❖ For loop

```
#include <iostream>

int main() {
  int i;

  for (i = 0; i < 10; i++) {
    std::cout << i << std::endl;
  }
  return 0;
}
```

C Style Support in C++

❖ While loop

```
while(Number<UpperLimit) //Number is less than UpperLimit.
{
    Number=Number+1; //Increment number one by one.
    sum=sum+Number; //All numbers will added to sum and will store in sum variable.
    //Example. sum=0+1=1; sum=1+1=2 and so on.
}
```

❖ if-else statement

```
if(percentage >= 60)
  {
     //if condition is true then show this:
     cout<<" Congratulation You are pass: "<<endl;
  }

  else
  {
      //if condition is false then show this:
     cout<<" you are fail: "<<endl;
  }
```

C Style Support in C++

❖ Switch statement

```
#include <iostream>
using namespace std;
main()
{
    char grade; //used to store Grade from user.

    cout<<" Enter Your Grade: ( Only A, B, C, and D grades are available:) ";
    cin>>grade;

    switch (grade)
    {
        case 'A': //read note to know about these types of case.
        case 'a':
        cout<<" Excellent...!Keep it up:\n ";
        break;

        case'B':
        case'b':
        cout<<" Good...! Need more care \n";
        break;

        case'C':
        case'c':
        cout<<" fair...! Need to be careful about your study: \n";
        break;

        case'D':
        case'd':
        cout<<" Fail...!!call your parents tomorrow.:\n ";
        break;

        default:
        cout<<" Only A, B, C, and D grades are avaliable. \n";
    }

    return 0;
}
```

OOP with C++

Reference

What is Reference?

- Reference defines an alternative name for an object

/*With a pointer*/

```
#include <iostream>

int change_val(int *p) {
  *p = 3;

  return 0;
}

int main() {
  int number = 5;

  std::cout << number << std::endl;
  change_val(&number);
  std::cout << number << std::endl;
}
```

/*With a reference*/

```
#include <iostream>

int change_val(int &p) {
  p = 3;

  return 0;
}
int main() {
  int number = 5;

  std::cout << number << std::endl;
  change_val(number);
  std::cout << number << std::endl;
}
```

Declaration and Initialization

❖ Reference must be initialized when declared

```
int a = 1024;
int &ref = a;          // ok
int b = 3;             //
ref = b;               // equal to a = 3;


ref &= b;
```

Totally different meaning!
(& operation)

```
int &ref;              //Wrong why?
```

Pointer vs. Reference

❖ What's the difference?

```
int num = 1024;
int& ref = num;
int* p = &num;

ref++;          // 1024 + 1
p++;            //maybe error
```

Reference to Reference

❖ What are the values of x, y, z?

```
int x;
int& y = x;
int& z = y;

x = 1024;
```

Reference with rvalue

- lvalue and rvalue
  - lvalue (pronounced “ell-value”)
    - Can stand on the left-hand side of an assignment
    - Can have address in memory
  - rvalue (pronounced “are-value”)
    - Cannot stand on the left-hand side of an assignment
    - Temporal value
    - Only can have a const reference

```
int &ref = 4;          //Error
const int &ref = 4;    //ok

int a = ref;          // equal to a = 4;
```

const qualifier: defines the variable is not changeable

Reference with Arrays

❖ Array of reference

```
int a, b;
int& arr[2] = {a, b};      // This is not allowed in C++
```

❖ Reference to an Array

```
int a[4] = {1,2,3,4};
int (&ref)[4] = a;

ref[0] = 10;
ref[1] = 20;

int b[2][2] = {1,2,3,4};
int (&ref2)[2][2] = b;
```

Allowed and valid operations!

Reference with Function

```
#include <iostream>

int func(int &a) { return a; }

int main() {
    int x = 1;
    std::cout << func(x)++ << std::endl;
}
```

Compile error!
Func returns 'rvalue', which can not be incremented.

```
#include <iostream>

int& func(int &a) { return a; }

int main() {
    int x = 1;
    std::cout << func(x)++ << std::endl;
    std::cout << x << std::endl;
    getchar();
}
```

This is ok ☺

Reference with Function

- Example

```
#include <iostream>

class A {
    int x;

public:
    A(int c) : x(c) {}

    int& return_ref_x() { return x; }
    int return_x() { return x; }
    void status_x() { std::cout << x << std::endl; }
};

int main() {
    A a(5);
    a.status_x();

    int& c = a.return_ref_x();
    c = 2;
    a.status_x();

    int d = a.return_ref_x(); // Copy of value
    d = 1;
    a.status_x();

    // Error. The value returned from return_x is rvalue.
    // int& err = a.return_x();
    // err = 2;
    // a.status_x();

    int f = a.return_x();
    f = 1;
    a.status_x();

    getchar();
}
```

A::x = 5

A::x = 2

A::x = 2

A::x = 2

Reference with Function cont'd

- Example

```
#include <iostream>

class A {
    int x;

public:
    A(int c) : x(c) {}

    int& return_ref_x() { return x; }
    int return_x() { return x; }
    void status_x() { std::cout << x << std::endl; }
};

int main() {
    A a(5);
    a.status_x();

    int& c = a.return_ref_x();
    c = 2;
    a.status_x();

    int d = a.return_ref_x(); // Copy of value
    d = 1;
    a.status_x();

    // Error. The value returned from return_x is rvalue.
    // int& err = a.return_x();
    // err = 2;
    // a.status_x();

    int f = a.return_x();
    f = 1;
    a.status_x();

    getchar();
}
```

```
26  // Error. The value returned from return_x is rvalue.
27  const int& err = a.return_x();
28  // err = 2;
29  a.status_x();
30
```

References

- *geeksforgeeks.org*
- *modoocode.com*
- *www.learncpp.com*
- *sourcemaking.com*
- www.infobrother.com
- *Lecture Slides for Programming in c++, google books*