In [1]:

```
%load_ext cppmagic
```

Introduction to C++

Andrew Kubera

Nov 19, 2014

C++

- Object oriented programming language, extending C
    - Most C programs are valid C++ programs
- Developed by Bjarne Stroustrup at Bell Labs ~1979
- The plus-plus represents the "iteration" of C
- Still evolving : new standards C++11/C++14

Keywords

C

> int double float short long char struct enum union void signed unsigned
>
> auto const extern static volatile
>
> if else switch case
>
> for while do continue break
>
> default goto register sizeof typedef return

count : 32

C++

> int double float short long char struct enum union void signed unsigned
>
> auto const extern static volatile
>
> if else switch case
>
> for while do continue break
>
> default goto register sizeof typedef return

> bool true false wchar_t
>
> class namespace this
>
> new delete
>
> try catch throw
>
> public protected private
>
> dynamic_cast reinterpret_cast const_cast static_cast
>
> asm explicit mutable typeid operator template typename friend using inline virtual

count : 62

python

> if elif else
>
> return yield try except finally raise
>
> and or not is in with as
>
> import from
>
> for while continue break
>
> del class def lambda
>
> return global assert pass print exec

count : 32

Compiling

GNU

- gcc
- g++

Clang/LLVM

- clang
- clang++

ROOT (CINT)

- root <filename>+

C Example

In [2]:

```
%%cpp
#include <stdio.h>
int main() {
  puts("Hello World");
  return 0;
}
```

```
Hello World
```

C++ Example

In [3]:

```
%%cpp
#include <cstdio>

int main() {
  for (int i = 0; i < 3; i++)
  puts("Hello World");
  return 0;
}
```

```
Hello World
Hello World
Hello World
```

Pointers

- Contain memory location of a variable.
- Created with: `TYPENAME *ptr`.
- Get the memory location with: `&VARIABLE_NAME`
- Get value pointed at with: `*POINTER_NAME`.

```
int a = 7;
int *ptr = &a;     // ptr 'points' to a
cout << a;         // 7
cout << ptr;       // memory location (0x7fff5500e792)
cout << *ptr;      // 'dereference' a (= 7)
*ptr = 12;         // 'a' now equals 12
cout << a;         // 12
```

In [6]:

```
%%cpp
#include <iostream>
using namespace std;
int main() {
  int a = 7, b = 9;
  int *ptr = &a;
  cout << a << '\n';
  cout << ptr << '\n';
  cout << *ptr << '\n';
  *ptr = 12;
  cout << "a == " << a << "\n";
  return 0;
}
```

```
7
0x7fff5b5b876c
7
a == 12
```

References

- Steals the identity of another variable
- Does not allocate new space
- Not a pointer, but CAN change the value of another variable

In [8]:

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

void add_three(int& x) {
  x += 3;
  cout << "X is at location: " << &x << "\n";
}

int main() {
  int a = 7;
  cout << a << '\n';
  add_three(a);
  cout << a << '\n';
  cout << "a is at location: " << &a << "\n";


}
```

```
7
X is at location: 0x7fff51ad274c
10
a is at location: 0x7fff51ad274c
```

Memory Management

- `C : malloc(), free()` ← Standard System Calls

```
int *a = malloc(sizeof(int) * 100);
...
free(a);
```

- `C++: new, delete` ← Language keywords

```
int *i = new int; // create integer in dynamic memory space
                  // (the 'heap')

*i = 1025;  // Set the value at memory position 'i' to 1025

int *a = new int[100]; // create 100 integers -
                       // size is figured out for you!

*(a + 3) = 8; // Set the third value from memory
              // position 'a' to 8
a[3] = 4;     // Set the same value to 4
a[0] = 31415; // Set the ZEROTH value to 31415

delete i;
delete[] a;
```

In [26]:

```
%%cpp
#include <iostream>
using namespace std;
int main()
{
  int *a = new int;
  cout << "a : " << a << ' ' << *a << "\n";
  *a = 100;
  cout << "a : " << a << ' ' << *a << "\n";

  delete a;
  return 0;
}
```

```
a : 0x7ffcf8403a60 0
a : 0x7ffcf8403a60 100
```

More C/C++ Differences

Functions can have the same name with different arguments

In [9]:

```
%%cpp
#include <cstdio>
using namespace std;

void foo(int a) {
  printf("int: %d\n", a);
}

void foo(float a) {
  printf("float: %f\n", a);
}

void foo(double a) {
  printf("double: %f\n", a);
}

int main() {
    foo(10);
    foo(10.);
    foo(10.f);
    return 0;
}
```

```
int: 10
double: 10.000000
float: 10.000000
```

Functions must be declared BEFORE use

In [17]:

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

int main()
{
  printstuff();
  return 0;
}

void printstuff() {
  cout << "cout  : " << std::scientific << 88923749827.234 << " : " << 88
923749827.234 << "\n";
  printf("printf: %e : %f ", 88923749827.234, 88923749827.234);
}
```

```
_cpp_magic_157df8d3ecfb6ee5eeff307ab1fd6843.cpp: In function 'int main()':
_cpp_magic_157df8d3ecfb6ee5eeff307ab1fd6843.cpp:8:14: error: 'printstuff'
was not declared in this scope
   printstuff();
              ^

ERROR: command `g++ _cpp_magic_157df8d3ecfb6ee5eeff307ab1fd6843.cpp -o _cp
p_magic_157df8d3ecfb6ee5eeff307ab1fd6843.o` failed.
```

Object Oriented Programming

What/Why?

- Group data and functions together into classes.
- Create objects from classes
    - Each with their own version of the data
    - The object has the class as its 'type'
- The object's functions retreive/manipulate the data

- Classes can inherit data/functions from 'parent' classes

Object Oriented Programming

Principles

- Encapsulation
    - Objects 'hide' data from rest of program ("members")
    - Only *objects themselves* can change their state through use of functions or 'methods'

- Inheritance
    - Classes derived from other classes
        - Reuse classes
        - Copy and extend functionality
    - Start with general classes and become more specific
    - Less programming is required when adding functions to complex systems

- Polymorphism
  - Different types can 'behave' the same
    - Printing out a description
    - Finding area of a shape ::
      - `area(circle)`
      - `area(rectangle)`
    - Multiplication:
      - `scalar * scalar`
      - `vector * vector`
      - `scalar * vector`
  - Child classes may behave as their parent class

Streams

- C++ provides 3 standard 'console' streams
  - `#include <iostream>`
  - `std::cout` - standard output
  - `std::cin` - standard input
  - `std::cerr` - error output
- Streams are a good example of **polymorphism** by printing of different types with same interface.
- Different kinds of streams
  - File streams : `#include <fstream>`
  - String streams : `#include <sstream>`
- Use new 'stream operators': >> and <<
  - Think : things flowing into and out of the objects

```
int *ptr;
std::cout << "one " << ptr << ' ' << *ptr << " " << 3;
int i;
std::cout << "Please type an integer: ";
std::cin >> i;

if (i < 100) {
  std::cerr << "Error, number too small!\n";
  exit(1);
}
```

In [42]:

```
%%cpp
#include <iostream>
#include <sstream>

int main()
{
  int i = 9;
  std::stringstream ss;
  ss << "one " << 2 << ' ' << 3.0 << " " << i << ' ' << &i << "\n";
  const char *ptr = ss.str().c_str();
  printf("%p %s\n", ptr, ptr );
  return 0;
}
```

```
0x7face0403d38 one 2 3 9 0x7fff5b49d76c
```

Templates

- Templates provide automatic polymorphism by demanding that functions or classes act on certain data types.
- Specify functions which will have types *later*
- Compile-time determination

In [49]:

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

template <typename T>
void foo(T x) {
  cout << "[foo] " << x << '\n';
}

void foot(char x) {
  cout << "[foot] " << x << '\n';
}

int main() {
foo<char*>(78);
 foot(78);
 return 0;
}
```

```
_cpp_magic_aa35fa4dcb3b3ed5f9e8ae1ceef79b3a.cpp: In function 'int main()':
_cpp_magic_aa35fa4dcb3b3ed5f9e8ae1ceef79b3a.cpp:15:14: error: no matching
function for call to 'foo(int)'
 foo<char*>(78);
              ^
_cpp_magic_aa35fa4dcb3b3ed5f9e8ae1ceef79b3a.cpp:15:14: note: candidate is:
_cpp_magic_aa35fa4dcb3b3ed5f9e8ae1ceef79b3a.cpp:6:6: note: template<class
T> void foo(T)
 void foo(T x) {
      ^
_cpp_magic_aa35fa4dcb3b3ed5f9e8ae1ceef79b3a.cpp:6:6: note:   template argu
ment deduction/substitution failed:
_cpp_magic_aa35fa4dcb3b3ed5f9e8ae1ceef79b3a.cpp:15:14: note:   cannot conv
ert '78' (type 'int') to type 'char*'
 foo<char*>(78);
              ^

ERROR: command `g++ _cpp_magic_aa35fa4dcb3b3ed5f9e8ae1ceef79b3a.cpp -o _cp
p_magic_aa35fa4dcb3b3ed5f9e8ae1ceef79b3a.o` failed.
```

Standard Template Library

C++ comes with standard containers and algorithms which make a lot of things easier.

std::vector

- Not mathematical vectors!
- Automatically allocates memory and stores whatever type is in template
    - Better than an array
- Fill with `push_back()`/`insert()`
- Good for random access elements

```
#include <vector>

std::vector<int> vi; // vector of integers
std::vector<float> vf; // vector of floats
std::vector<std::vector<int> > vv; //vector of int vectors
```

In [18]:

```
%%cpp
#include <vector>
#include <iostream>

int main()
{
   std::vector<int> *v = new std::vector<int>();
   for (int i = 0; i < 200000; i++) {
     v->push_back(i * 10);
   }
  std::cout << "Size : " << v->size() << "\n";
  std::cout << "v[19] : " << (*v)[9] << '\n';
  delete v;
  return 0;
}
```

```
Size : 200000
v[19] : 90
```

In [57]:

```
%%cpp
#include <vector>
#include <iostream>

int main()
{
  // CONSTRUCTOR :: Vector has 20 elements with the number '2'
  std::vector<float> v(20);
  std::cout << "Size : " << v.size() << "\n";
  std::cout << "v[19] : " << v[19] << '\n';
  std::cout << "-----\n";
  for (int i = 0; i < 20; i++) {
     v.push_back(i * 10.5);
  }
  std::cout << "Size : " << v.size() << "\n";
  std::cout << "v[29] : " << v[29] << '\n';
  return 0;
}
```

```
Size : 20
v[19] : 0
-----
Size : 40
v[29] : 94.5
```

std::string

- Automatic
  - size allocation
  - concatenation
  - comparison

```
#include <string>

std::string s("Hello"); // Create a string with the char[] "Hello";
s += " world";          // Add " " to the end of string 's'
std::cout << s << " :-D "

if (s == "Hello world") {
 ...
}
```

In [19]:

```
%%cpp
#include <string>
#include <iostream>

int main() {
 std::string s("Hello"); // Create a string with the char[] "Hello";
 s += " world";          // Add " " to the end of string 's'
 std::cout << s << " :-D\n";
 if (s != "Hello world") {
   std::cout << "EQUAL!\n";
 }
 std::cout << "substr: " << s.substr(3, 8) << "\n";
 std::cout << s.substr(s.find('o')) << "\n";
 return 0;
}
```

```
Hello world :-D
substr: lo world
o world
```

std::map

- Associative array, requires 2 template types

```
std::map<int, std::string> m; // Maps numbers to strings
m[17] = "Seventeen";
```

In [22]:

```
%%cpp
#include <string>
#include <map>
#include <iostream>
int main() {

std::map<int, std::string> m; // Maps numbers to strings

m[17] = "Seventeen";
m[1] = "One";
m[42] = "The answer to life the universe and everything";
m[2] = "The Only even prime";


std::cout << "m.size() == " << m.size() << "\n";
std::cout << "m[17] = '" << m[17] << "'\n";
std::cout << "m[2]  = '" << m[2] << "'\n";
return 0;
}
```

```
m.size() == 4
m[17] = 'Seventeen'
m[2]  = 'The Only even prime'
```

Many More!

- std::list
- std::set
- std::queue
- std::multimap
- std::pair

- std::complex
- std::random

Classes

- Grouped data (members) and functions (methods)
- All functions have access to data
- Data hiding-
  - `public` : Everybody has access
  - `protected` : Children have access
  - `private` : Only you have access

```
class CLASSNAME {
public:
  CLASSNAME();      // Constructor
  CLASSNAME(int x); // Constructor

  void SayHi();      // Method
  int GetX() const;  // Constant Method

private:
  int _x; // Private Member
};
```

- Define functions with double colon

```
int CLASSNAME::GetX() {
  return _x;
}
```

- Access the 'current' object using the `this` keyword
  - `this` is a pointer to the current object, and can be treated as such

```
void SayHi() {
   std::cout << "the object at " << this << " says hi.\n";
}
```

In [66]:

```
%%cpp
#include <iostream>

// Class Declaration
class A {
public:
  A();
  void SayHello();
  int _count; // Counts number of times 'A' says "hello"
};

// Constructor Definition
A::A():
 _count(0) // initialize _count to zero
{

}

void A::SayHello() {
  std::cout << "HELLO! (" << _count++ << ", "<< this <<") \n";
}

int main() {
 A a;
 a._count = 20;
 a.SayHello();
 a.SayHello();
 a.SayHello();
 A b;
 b.SayHello();
 a.SayHello();
 return 0;
}
```

```
HELLO! (20, 0x7fff5485b780)
HELLO! (21, 0x7fff5485b780)
HELLO! (22, 0x7fff5485b780)
HELLO! (0, 0x7fff5485b770)
HELLO! (23, 0x7fff5485b780)
```

Inheritance

- Children get `public`/`protected` members/methods of parent class
- Specify parents at class declaration

```
class Child : public Parent {

...

};
```

Multiple Inheritance

- Can inherit members from multiple classes

```
class Child : public Parent1, public Parent2 {

...

};
```

In [24]:

```
%%cpp
#include <iostream>

class A {
public:
  A() {
   std::cout << "Constructing A @" << this << "\n";
  }

  void Print() {
    std::cout << "I am an 'A'\n";
  }
};

class B : public A {
public:
  B() {
   std::cout << "Constructing B @" << this << "\n";
  }
};

int main() {
 A a;
 B b;

 std::cout << "---\n";
 a.Print(); // A::Print()
 b.Print(); // A::Print()

//# A *c = (A*)&b;
//# c->Print();

 return 0;
}
```

```
Constructing A @0x7fff4ff8a74f
Constructing A @0x7fff4ff8a74e
Constructing B @0x7fff4ff8a74e
---
I am an 'A'
I am an 'A'
```

Abstract Classes

- Sometimes you don't want to make an object from a class
  - Force developers to subclass
- Use 'Pure Virtual Functions' to do this

```
class Shape {
public:
 Shape();                  // constructor
 virtual double Area()=0; // 'pure virtual' function
};
```

- User must subclass Shape to use it

```
class Circle : public Shape {
public:
    Circle(double radius): _r(radius){};

    double Area() {
      return 3.1415 * _r * _r;
    }

protected:
   double _r;
};
```

In [25]:

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

class Shape {
public:
    Shape(){};  // constructor
    virtual double Area() = 0; // 'pure virtual' function
};

int main()
{
  Shape s;
  return 0;
}
```

```
_cpp_magic_331ad67cd32b82d92801e90cda4050ad.cpp: In function 'int main()':
_cpp_magic_331ad67cd32b82d92801e90cda4050ad.cpp:12:9: error: cannot declar
e variable 's' to be of abstract type 'Shape'
   Shape s;
         ^
_cpp_magic_331ad67cd32b82d92801e90cda4050ad.cpp:4:7: note:   because the f
ollowing virtual functions are pure within 'Shape':
 class Shape {
       ^
_cpp_magic_331ad67cd32b82d92801e90cda4050ad.cpp:7:20: note:     virtual do
uble Shape::Area()
     virtual double Area() = 0; // 'pure virtual' function
                    ^

ERROR: command `g++ _cpp_magic_331ad67cd32b82d92801e90cda4050ad.cpp -o _cp
p_magic_331ad67cd32b82d92801e90cda4050ad.o` failed.
```

In [26]:

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

class Shape {
public:
  Shape(){};                 // constructor
  virtual double Area() = 0; // 'pure virtual' function
};

class Circle : public Shape {
public:
    Circle(double radius): _r(radius){};

    double Area() {
      return 3.1415 * _r * _r;
    }

protected:
   double _r;
};

class Rectangle : public Shape {
public:
    Rectangle(double length, double width): _l(length), _w(width)
    {
     cout << "Created rectangle with length=" << _l
                             << " and width=" << _w << "\n";
     };

    double Area() {
      return _l * _w;
    }

protected:
   double _l;
   double _w;
};

void PrintArea(Shape& s)
{
  std::cout << "area == " << s.Area() << endl;
}

int main()
{
  Rectangle r(5,10);
  Circle c(25.);

  PrintArea(r);
  PrintArea(c);

   cout << "---\n" << "area == " << r.Area() << endl;

  return 0;
}
```

```
Created rectangle with length=5 and width=10
area == 50
area == 1963.44
---
area == 50
```

In [ ]:

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

class Shape {
public:
  Shape(){};                 // constructor
  virtual double Area() = 0; // 'pure virtual' function
};

class Circle : public Shape {
public:
    Circle(double radius): _r(radius){};

    double Area() {
      return 3.1415 * _r * _r;
    }

protected:
   double _r;
};

int main()
{
  Circle c(25.);

  cout << "Circle area == " << c.Area() << endl;
  return 0;
}
```

Operators

- Special functions in the class which allow programmers to use symbols like '+', '-'
- `operatorXXX()`, where XXX is the symbol

```
class A {
protected:
 int _x;                 // protected integer _x
public:
  A():_x(0){};           // Default _x = 0
  A(int i): _x(i) {};    // Construct with _x

  A operator+(A& a) {    // ADDITION OPERATOR
    return A(_x + a._x); // A+A
  }

  A operator+(int i) {   // ADDITION OPERATOR
    return A(_x + i);    // A + int
  }

  A& operator+=(int i) { // ADDITION/ASSIGNMENT
    _x += i;             // A += int
    return *this;
  }

  void Print() {
    std::cout << "My _x == " << _x << "\n";
  }

};
```

In [31]:

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

class A {
public:
  A():_x(0){};
  A(int i): _x(i) {};


  A operator+(A& a) {
    return A(_x + a._x);
  }

  A operator+(int i) {
    return A(_x + i);
  }

  void operator+=(int i) {
    _x += i;
  }

  void Print() {
    std::cout << "My _x == " << _x << "\n";
  }

protected:
 int _x;
};

int main()
{
  A a;
  cout << "a: "; a.Print();
  a += 4;
  cout << "a: "; a.Print();
  A b(10), c = a + b;
  cout << "b: "; b.Print();
  cout << "c: "; c.Print();
  return 0;
}
```

```
a: My _x == 0
a: My _x == 4
b: My _x == 10
c: My _x == 14
```

In []: