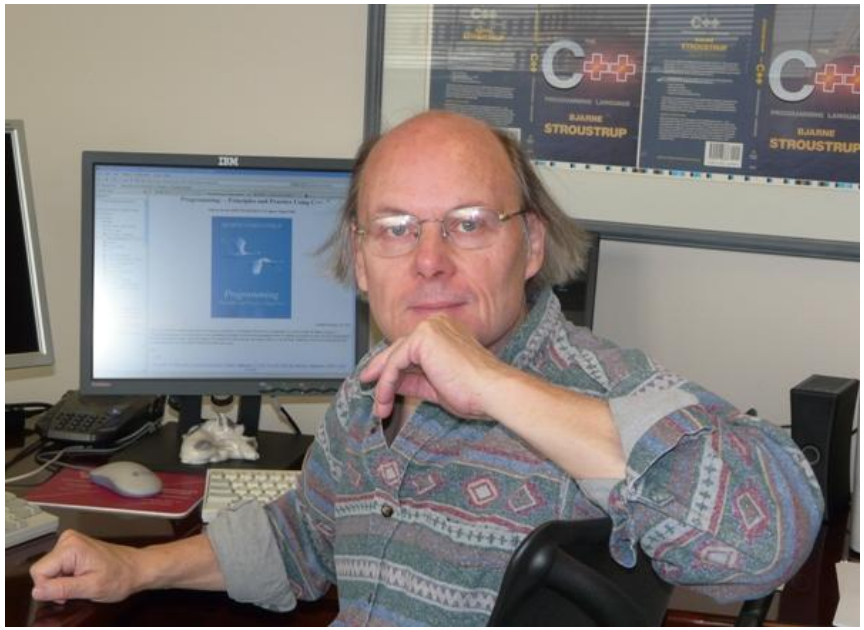
C++

Ben Borchard, Terrence Tan, Scott Franchi, Joseph Harwood, James Staley

History

Bjarne Stroustrup



C with classes

http://www.stroustrup.com/

Classes

- Defined with private and public fields and methods
- Methods can be defined in class definition or later

```
class Werewolf{
    int humansChanged;
    float bloodThirst;
    string wolfName;
    string getName();
public:
    static int totalKills;
    Werewolf();
    ~Werewolf(); //destructor method
    Werewolf(int hC, float bT, string wN){
        humansChanged = hC;
        bloodThirst = bT;
        wolfName = wN;
        ancientKnoweldge = "Don't let anyone know you're a werewolf";
    };
    void info();
    float getBloodThirst();
    Werewolf* operator + (Werewolf);

//Protected variables are accessable by any class which inherits Werewolf
protected:
    string ancientKnoweldge;

} wallace; //Instances of the class created with the default constructor
```

```cpp
//intialize static variable
int Werewolf::totalKills = 0;

Werewolf::Werewolf(){
    humansChanged = 0;
    bloodThirst = 0.0;
    wolfName = "Wallace";
    ancientKnoweldge = "The tastiest humans are the children";
}

//destructor method called when the object is deleted, destructors are usefull for freeing dynamically allocated memory
Werewolf::~Werewolf(){
    cout << wolfName << " has died" << endl;
}

string Werewolf::getName(){
    return wolfName;
}

float Werewolf::getBloodThirst(){
    return bloodThirst;
}

void Werewolf::info(){
    cout << wolfName << " has changed " << humansChanged << " humans into werewolves" << endl;
}

//Operators can be overloaded to perform unique operations on classes
Werewolf* Werewolf::operator+ (Werewolf parent){
    Werewolf* baby = new Werewolf();
    return baby;
}

int main(){
    wallace.info();
    Werewolf *hector = new Werewolf(5, 1.2, "Hector");
    hector->info();
    Werewolf* babyWolf = *hector + wallace;

    delete hector;
    return 0;
}
```

Functions

- C++ supports function overloading and default parameters, unlike C

```
int testfunc(int opt=5){
    cout<< "Printing argument: "<< opt<<endl;
    return 0;
}
int testfunc(int a, int b){
    return a*b;

}

int main(){
    testfunc();
    cout<< testfunc(5,4)<<endl;
    return 0;
}
```

Type Casting

- Standard C casts still function properly
- Implicit type conversions can reduce the need for explicit casts
- New casting functions enable safer casts
  - dynamic_cast
  - static_cast
  - reinterpret_cast
  - const_cast
- The typeid function can compare class types

IOstream

```
#include <iostream>

using namespace std;

int main(){
    string s;

    cout << "hello world, how are you today?\n";
    getline(cin, s);
    cout << "very cool, I am also " << s << '\n';
    return(0);
}
```

File IO

C

```c
#include <stdio.h>
#include <stdlib.h>

int main(){
    int i;
    char l[100];

    FILE *file1 = fopen("okeydokey.txt", "w");
    fprintf(file1, "OkeyDokeySmokey\nFuzzie Wuzzie\n");
    fclose(file1);

    FILE *file2 = fopen("okeydokey.txt", "r");

    fseek(file1, 0, SEEK_SET);

    for (i=0; i<2; i++) {
        fscanf(file1, "%s\n", &l[0]);
        printf("%s\n", l);
    }
    fclose(file2);

    return (0);
}
```

C++

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

using namespace std;

int main(){
    string l;

    int a = sizeof(string);

    ofstream f1;
    f1.open ("okeydokey.txt");
    f1 << "OkeyDokeySmokey\n" << "FuzzieWuzzie\n";
    f1.close();

    ifstream f3;
    f3.open("okeydokey.txt");
    while (f3.good()){
        getline (f3, l);
        cout << l << endl;
    }
    f3.close();

    return (0);
}
```

Templates

- C++ allows for polymorphism through the use of templates to declare generic types
- Both classes and functions can be templated
- Two ways of creation:

```
template <class T>
```

```
template <typename T>
```

- Both of these have the same meaning and behave in the same way
- Fill in templates when calling the class or function.

```
LinkedList <int> *lnk = new LinkedList<int>();
```

- Templates within classes are placed above each method

```
template <class T>
T LinkedList<T>::llpop(){
```

Templates continued

- Templates can be specialized so that a templated class will act differently for a specific type

```
template <>
class LinkedList <int>{
 //some function only for when int is the given data type
}
```

- Can assign default values for templates much like default parameters
- Can declare multiple templated types within one template

```
template <class T = int, typename T2 = char>
```

Differences from C

- Function overloading
- Casting conventions
- Boolean Types
- Default arguments
- Error Handling
- File I/O
- Strings
- Classes
- Namespace
- IOstream

Key Words

- Main differences from C: explicit, class, new, friend, delete, inline, this, protected, private, public, try, catch, typeid and the 4 cast types mentioned earlier.
- Most of these come along with the transition from non-object oriented to object oriented (friend, class, template, this, public ...)
- You can see too that error catching/casting makes up most of the other differences

Why use C++...

- Over C
  - has additional libraries to support general purpose programming
  - Supports exception handling
  - Supports multiple programming paradigms.
- Over Java
  - greater control over memory (explicit memory management)
  - compatible with C code
  - No overhead from JVM

Sources

http://www.stroustrup.com/
http://www.cplusplus.com/info/history/
http://www.cplusplus.com/doc/tutorial/templates
http://www.cplusplus.com/doc/tutorial/functions2/
http://www.cplusplus.com/doc/tutorial/typecasting/