Developing software means that we have to think in our Stack of technologies to provide a solution.

So one use for C++/C is to build libraries depending the OS that we have, But there is a weakness in C++, depending the compiler you used, maybe your application is not BINARY COMPATIBLE.

A solution to this problem for example is exposing a small C interface from your C++ code, in order to achieve this compatibility.

int \_tmain(int argc, \_TCHAR\* argv[])

{

auto a = Item();

auto b = Item("Number");

a.hello();

b.hello();

return 0;

}

#include <iostream>

#include <string>

using namespace std;

/\* Definition simple class \*/

class Item {

private:

string type;

public:

Item();

~Item();

Item(string name);

virtual void hello();

};

/\* Implementation \*/

Item::Item() : Item("None") {}

Item::~Item(){}

Item::Item(string name) {

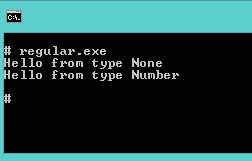
this->type = name;

}

void Item::hello(){

cout << "Hello from type " << type << "\n";

}



To declare a function to be compatible with c and use the name convention, we must use the extern "C" {} clause. example:

void myFuntion(){

cout << "Hello from my function\n";

}

extern "C" {

void myFoo(){

cout << "This is a classic c function\n";

}

}