Type-safe Linkage

Consider the following program:

```
// test.cpp
#include <iostream>
using namespace std;

void print(int n) {
  cout << "int: " << n << endl;
}

void print(char ch) {
  cout << "char: " << ch << endl;
}

void print(unsigned char ch) {
  cout << "unsigned char: " << ch << endl;
}

void print(float f, int n = 1) {
  cout << f << ' ' << n << endl;
}

int main() {
  print('a');
}
```

We'll look at the symbols in the object file created by g++ (version 4.1.2):

```
$ g++ -c test.cpp
$ nm test.o
00000044 t _GLOBAL__I__Z5printi
00000000 t _Z41__static_initialization_and_destruction_0ii
000000be T _Z5printc
00000074 T _Z5printfi
00000172 T _Z5printh
00000134 T _Z5printi
         U _ZNSolsEPFRSoS_E
         U _ZNSolsEf
         U _ZNSolsEi
         U _ZNSt8ios_base4InitC1Ev
         U _ZNSt8ios_base4InitD1Ev
         U _ZSt4cout
         U _ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_
00000000 b _ZSt8__ioinit
         U _ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc
         U _ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_c
         U _ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_h
         U __cxa_atexit
         U __dso_handle
         U __gxx_personality_v0
0000005c t __tcf_0
00000108 T main
```

- function names have been "mangled"
- the manner of mangling is compiler-dependent

Quesion: How can we "call" a C function from a C++ program if names are mangled in C++, but not in C?

Answer: Use `extern "C"` when declaring a C function in your C++ program

Example

```
/* square.c (C source file)
   - compile with: gcc -c square.c  */
double square(double d) {
  return d * d;
}

// main.C (C++ source file)
// - compile with: g++ -c main.C
#include <iostream>
using namespace std;

extern "C" double square(double);

int main() {
  cout << square(1.2) << endl;
}
```

Without `extern "C"`, `square` would be mangled in the output from the C++ compiler whereas it is not mangled in the output from the C compiler; hence we won't be able to link them

In the above, we could also use

```
extern "C" {
  double square(double);
}
```

or, if the prototype is in a header named `square.h`,

```
extern "C" {
  #include "square.h"
}
```

We look at the object files:

```
square.o:
00000000 T square

main.o (without extern "C"):
000000aa t _GLOBAL__I_main
00000054 t _Z41__static_initialization_and_destruction_0ii
         U _Z6squared
         U _ZNSolsEPFRSoS_E
         U _ZNSolsEd
         U _ZNSt8ios_base4InitC1Ev
         U _ZNSt8ios_base4InitD1Ev
         U _ZSt4cout
         U _ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_
00000000 b _ZSt8__ioinit
         U __cxa_atexit
         U __dso_handle
         U __gxx_personality_v0
00000092 t __tcf_0
00000000 T main

main.o (with extern "C"):
000000aa t _GLOBAL__I_main
00000054 t _Z41__static_initialization_and_destruction_0ii
         U _ZNSolsEPFRSoS_E
         U _ZNSolsEd
         U _ZNSt8ios_base4InitC1Ev
         U _ZNSt8ios_base4InitD1Ev
         U _ZSt4cout
         U _ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_
00000000 b _ZSt8__ioinit
         U __cxa_atexit
         U __dso_handle
         U __gxx_personality_v0
00000092 t __tcf_0
00000000 T main
         U square
```