Integrating C with Other Languages

C Programming and Software Tools

N.C. State Department of Computer Science
Why integrate

• I want performance from C
• I want ease of use from higher level languages
• I want low-level control with assembly language

• Sometimes, I want all of these at once!

• Also, sometimes the library I need is only written for one language.
Language interactions

- Interpreted higher-level languages:
  - Perl
  - Python
  - PHP
- Compiled languages:
  - C
  - C++
  - Java
  - Pascal
  - FORTRAN
  - D
  - Go
- Platform-dependent machine code:
  - Assembly language
Most common language interactions

Interpretted higher-level languages:
- Perl
- Python
- PHP

Compiled languages:
- C#
- Java
- Ruby
- C++

Platform-dependent machine code:
- Assembly language

Most common case:
- Write in higher level language if you can
- Call assembly language if you need to
- Promote your program to C++ if you need to talk to C++ or do OOP
Attributes of language interaction

• What **direction**?
  – C calling other, or other calling C?

• What **mechanism**?
  – Direct call
  – Shim layer
    • Automatically generated or manually written?
  – Inlining foreign code
  – Other/weird (shared memory, common caller, etc.)

• Handling **language feature mismatches**?
  – E.g., garbage collection?
Language interactions

- **Interpreted higher-level languages**: Perl, Python, PHP
- **Compiled languages**: C, C++, D, Go, FORTRAN, Pascal
- **Platform-dependant machine code**: Assembly language

CSC230: C and Software Tools © NC State University Computer Science Faculty
Examples: Higher level languages

- **Direct**: Python’s built-in ctypes module can be used to create variables in C data types and to make calls to C shared libraries.
- **Shim (manually developed)**: Homework 6 includes a Python class that uses ctypes to wrap up calls to your libCTurtle.
- **Shim (auto-generated)**: The Perl tool h2xs generates stubs which call C library code based on a header file.
- **Inline**: The Perl Inline::C module is used to write C code directly mixed in with Perl code.

```bash
#!/usr/bin/perl
use Inline C;
hello_inline();
__END__
__C__
#include <stdio.h>

void hello_inline( ) {
    printf("Hello World inline!\n");
}
```
Examples: Higher level languages

- Calling other languages from C:
  - **Direct**: Java JNI allows you to run a JVM from C directly and run code on it.
  - **Shim (manually developed)**: You can write a C module using Java JNI.
  - **Shim (auto-generated)**: Java has a javah tool to create C header files from Java classes.
  - **Inline**: Python supports embedding in C via multiple interfaces, such as the Very High Level (VHL) interface.

```c
#include <Python.h>

int main(int argc, char *argv[])
{
    Py_SetProgramName(argv[0]);
    Py_Initialize();
    PyRun_SimpleString("from time import time,ctime\n"      "print 'Today is',ctime(time())\n");
    Py_Finalize();
    return 0;
}
```

Adapted from: https://docs.python.org/2/extending/embedding.html
Garbage collection

- When C is called from a garbage collected language, you hook the garbage collection notice to clean up C-based objects.
- This is done in the Homework 6 destructor:

```python
class CTurtle(object):
    c = ctypes.cdll.LoadLibrary("libCTurtle.so")
    c.get_last_error.restype = ctypes.c_char_p

    @classmethod
    def new(self, w, h):
        r = CTurtle()
        r.img = self.c.create_image(w, h)
        return r

    def __del__(self):
        self.c.destroy_image(self.img)
```
Language interactions

Interpreted higher-level languages:
- Perl
- Python
- PHP
- Ruby
- Java
- C#
- C++
- FORTRAN
- Pascal
- D
- Go

Compiled languages:
- C

Platform-dependant machine code:
- Assembly language
Examples: Other compiled languages

- **Direct**: You can link C and FORTRAN object files together
- **Inline**: n/a

```c
#include "my_sub.h"

int main(){
    double b;
    int a=3;
    my_sub(a,&b);
    ...
}
```

```fortran
MODULE my_fortran
    USE iso_c_binding
    IMPLICIT NONE
    CONTAINS
    SUBROUTINE my_subroutine(a,b) BIND(C,name="my_sub")
        INTEGER(c_int),INTENT(in),VALUE :: a
        REAL(C_DOUBLE),INTENT(out) :: b
        ...
    END SUBROUTINE
END MODULE
```

```c
void my_sub(int, double *);
```

Adapted from:
http://stackoverflow.com/questions/7743801/mixed-programming-fortran-and-c
Language interactions

Interpreted higher-level languages

Compiled languages

Platform-dependent machine code

C

Perl

C#

Python

PHP

Java

Ruby

C++

D

Go

FORTRAN

Pascal

C#

Java

PHP

Ruby

C++

D

Go

FORTRAN

Pascal

Assembly language
Interacting with assembly language

• Calling assembly from C:
  – **Direct**: Can call assembly routines in C directly (just compile them together)
  – **Shim**: You can wrap ‘em if you want.
  – **Inline**: C lets you put assembly right in your code (see next slide).

• Calling C from assembly:
  – **Direct**: Just need to put the arguments on the stack properly and do a call instruction
  – **Shim**: n/a
  – **Inline**: n/a
Inline assembly language

• Literally gets dumped in with the compiler-generated assembly instructors
• Useful for small tricks and recipes

What does this do?

```c
#include <string.h>

int main() {
    char* msg = "Testing!\n";
    asm (  
        "movq %1, %rdx\n" // set param 3 (length) to length of msg
        "movq %0, %rcx\n" // set param 2 (buffer) to address of msg
        "movq $1, %rbx\n" // set param 1 (file descriptor) to stdout (1)
        "movq $4, %rax\n" // set syscall number 4 (write)
        "int $0x80\n" // ask kernel to do it
    : // no outputs
    : "r"(msg), "r"(strlen(msg)) // inputs assigned to %0, %1, etc.
    : "%rax","%rbx","%rcx","%rdx" // tell compiler which registers we trashed
    );
}
```

$ gcc a.c && ./a.out
Testing!
Language interactions

Interpreted higher-level languages:
- Perl
- Python
- PHP
- C#
- Java
- Ruby

Compiled languages:
- C
- C++
- FORTRAN
- Pascal
- D
- Go

Platform-dependent machine code:
- Assembly language
Interacting between C and C++

• Letting C call C++
  – C++ is a thin layer of simple tricks on top of C which create OO-friendly syntax
  – To get C to call C++, you just need to cut through the tricks with `extern "C" { ... }`, which just says “disable C++ trickery for this part”.
    – See next slide

• Letting C++ call C
  – Just do it. No steps necessary.
C/C++ example

Duck.h

```c
#include "Duck.h"

class Duck {
public:
    Duck(int feet) : { ... }
    ~Duck() { ... }

    void quack(float volume) { ... }
};

struct Duck* new_Duck(int feet);
void delete_Duck(struct Duck* d);
void Duck_quack(struct Duck* d, float volume);
```

Duck.cpp

```c
extern "C" {
    #ifdef __cplusplus
    extern "C" {
    #endif

    struct Duck;

    struct Duck* new_Duck(int feet);
    void delete_Duck(struct Duck* d);
    void Duck_quack(struct Duck* d, float volume);

    #ifdef __cplusplus
    #endif
}
    #endif
```
Any Questions?