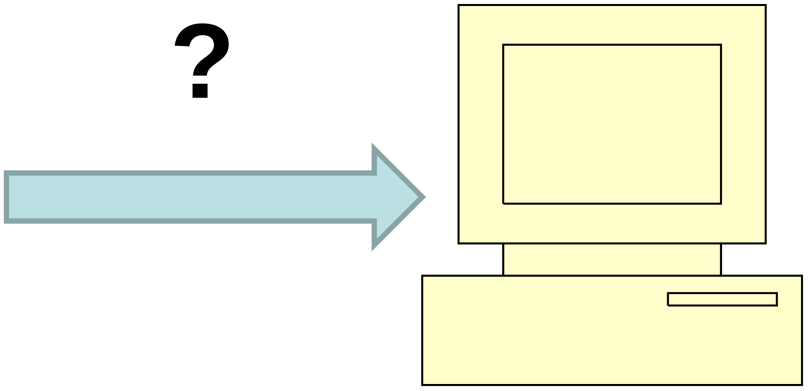
Building C Programs

Shawn T. Brown
Director of Public Health Applications
Pittsburgh Supercomputing Center

Computers do not understand programming languages…

```
#include <stdlib>
#include <stdio>

int main(){
   printf(“Hello Mom!”);
   return 0
}
```



Humans do not understand binary…

?

01001001001001010100100100100101010010010010010101001001001001010100100100101001001001001010100100100100101001001001001010100100100100101001010010010010010101001010010010010010100101001001001010100100100101001010010010010010101001001001001010100100100100100101010010010010010100100100100110010010010110

Humans do not understand binary…

010010010010010101001001001
001010100100100100101010010
010010010101001001001001010
100100100100101010010010010
010101001001001001010100100
100100101010010010010010101
001001001001010100100100100
101010010010010010101001001
001001010100100100100101010
010010010010101001001001001
010100100100100101010010010
010010101001001001001010100
10010010010110

Unless you are Axel!

For the rest of us…

- Programming languages have been created so that you do not have to write machine code.
- Generally speaking, programming languages are designed with specific requirements to translate something mere mortals can understand to machine code.
- Difficult, that is why it is not trivial to learn programming.

Computer Languages

- Generally a spectrum

PERL
Python
Java
PHP

Ease of use

Productivity

Rapid development

Performance

More control over computers resources

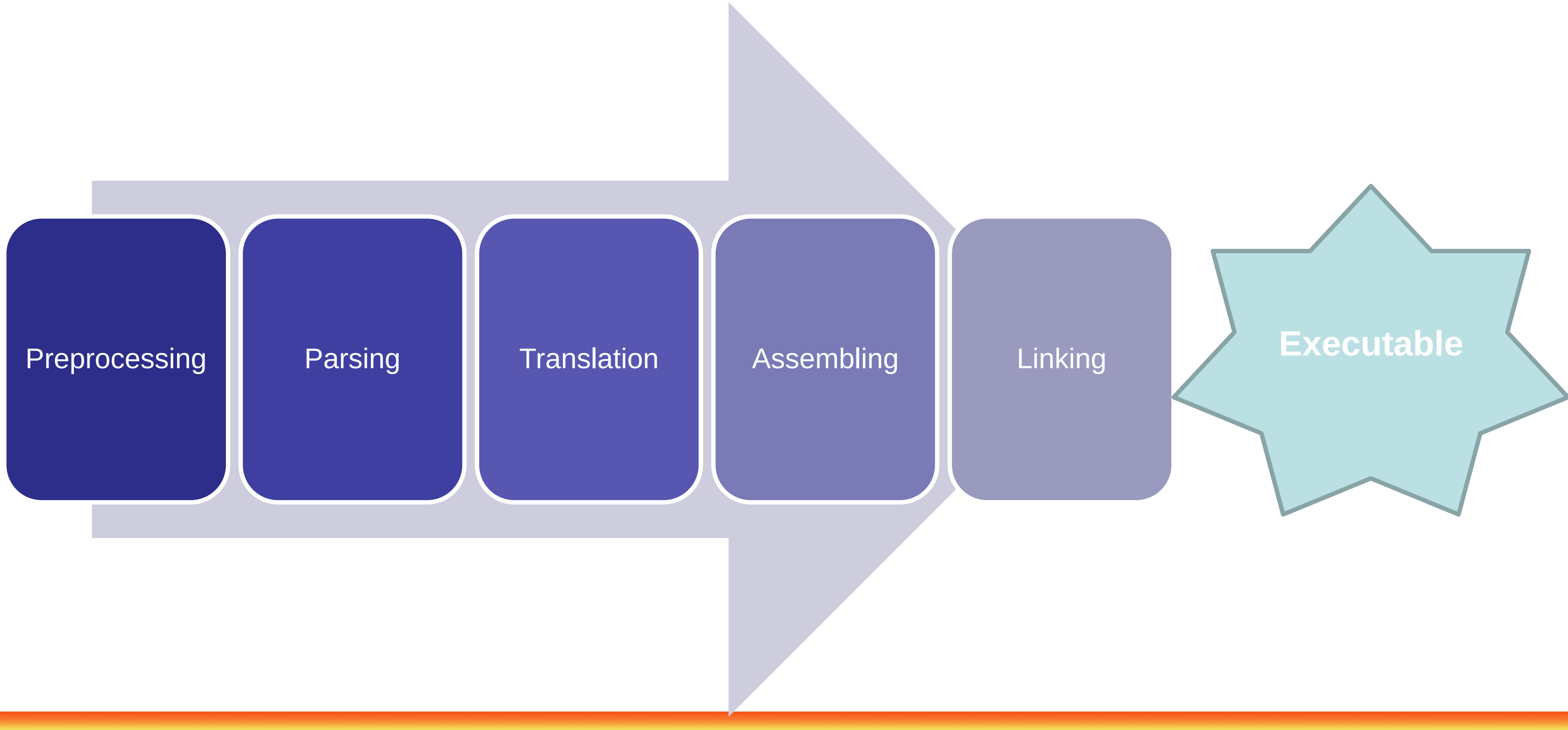
More complex

C
C++
FORTRAN

And then God gave us compilers…

- The compiler is the single most useful tool that a programmer has at his/her disposal.
- The compiler translates through a series of steps your “human-readable” source code to something the computer understands.
- All programming languages have to be compiled at some level.
  - In interpreted languages, this is done prior by another programmer that implements the interpreter on a given architecture.

Steps in a Modern Compilation Chain



Preprocessing

- This is the stage in the compilation where items such as directives
  - These are directives that can be defined in source (usually with a # before the line)
  - Can also be passed through the command line with –D
  - Basically just a substitution engine
  - gcc –E

Parsing and Translation

- This stage takes the preprocessed source files and translates them into some form of assembly language
- Optimization also happens in this phase
  - Automatic interpretation of common code constructs that can be rewritten in a more optimal manner (e.g. loop unrolling
- gcc -S

Assembly Stage

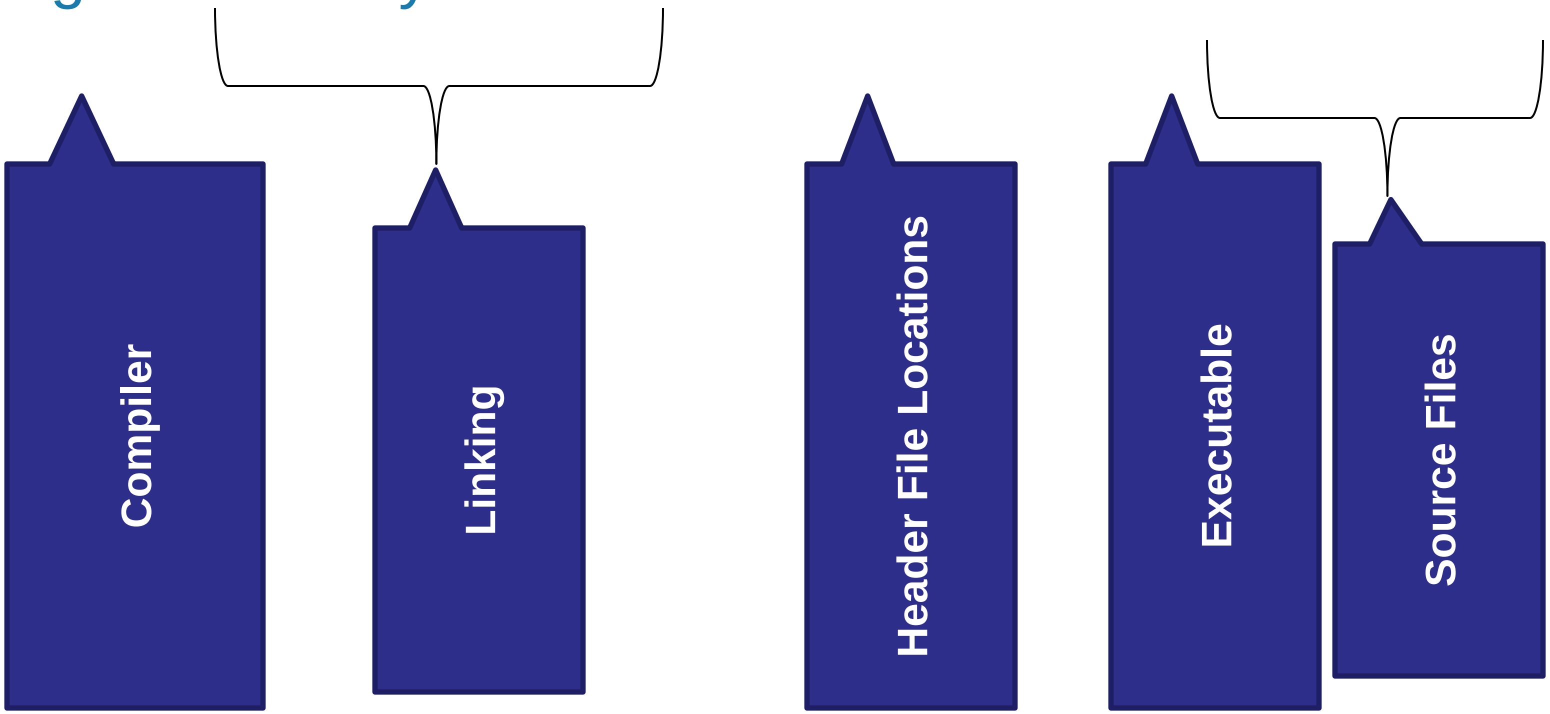
- Takes the assembly code and translates it to machine instructions
- Generally creates object files (.o) files for each source file given.

Linking Stage

- Linking takes and includes of the external libraries that are to be included in the executable.
- Usually defined with key words to the compiler like –lm (which specifies libmath)
- Static Linking:
  - Explicitly includes the libraries machine code into the executable (.a)
- Dynamic Linking:
  - Places a hook in executable that gets included at runtime (.so)

Basic Compilation command for C

- gcc –L/libraryDir –lm -I/includeDir–o foo foo.c bar.c



This syntax will suffice for most simple commands…. it actually runs through all of the compilation steps in one line.

Another way…

- If you would like more control over individual objects (different includes and libraries

  gcc –I/includeDir1 –c –o foo.o foo.c
- gcc –I/includeDir2 –c –o bar.o bar.c
- gcc –L/libDir1 –L/libDir2 -llib1 –llib2 –o foo foo.o bar.o

Makefiles

```
CC=gcc
CFLAGS=-I.
DEPS = hellomake.h

%.o: %.c $(DEPS)
	$(CC) -c -o $@ $< $(CFLAGS)

hellomake: hellomake.o hellofunc.o
	gcc -o hellomake hellomake.o hellofunc.o -I.

install: hellomake
	cp hellomake /usr/local/bin
clean:
	rm -f $(ODIR)/*.o *~ core $(INCDIR)/*~
```

With the make file, one just types “make” and the program compiles with all dependencies.

Advanced methods of compiling:

libtools – allows one to write makefiles that rely on a well defined set of architecture depend variables (this is what is used when you type ./configure)

Cmake – an platform independent tool chain for building source.

```c
#include <math.h>
#include <stdio.h>
#include "ctest.h"
#define NUM 5000000

float great_circle(float lon1, float lat1, float lon2, float lat2){
    float radius = 3956.0;
    float pi = 3.14159265;
    float x = pi/180.0;
    float a,b,theta,c;

    a = (90.0-lat1)*(x);
    b = (90.0-lat2)*(x);
    theta = (lon2-lon1)*(x);
    c = acos((cos(a)*cos(b)) + (sin(a)*sin(b)*cos(theta)));
    return radius*c;
}

int main() {
    int i;
    float x;
    for (i=0; i <= NUM; i++)
        x = great_circle(-72.345, 34.323, -61.823, 54.826);
    printf("%f\n", x);
}
```

Variable Scope

```
#include <stdio.h>

void foo(int a){
        a = 5;
        printf("a inside foo = %d\n",a);
}

int main(void){
        int a = 10;

        foo(a);
        printf("a = %d\n",a);
        return 0;
}
```

```
> ./test2
a inside foo = 5
a = 10
```

Pointers

- A pointer is a variable that holds the address to a location in memory.
- In C a pointer is signified by putting an “*” in front of the variable

```
#include <stdio.h>

int main(void){
int i = 1;
int *j = &i;
printf("I = %d j = %p *j = %d\n",i,j,*j);
return 0;
}

>gcc -o test test.c
>./test
>I = 1 j = 0x7fff16ef7fdc *j = 1
```

Passing variables to functions

```
#include <stdio.h>

void foo(int a, int *b){
        a = 5;
        *b = 9;
}
int main(void){

        int a = 2;
        int b = 3;

        foo(a,&b);
        printf("a = %d b= %d\n",a,b);
        return 0;
}
```

Passing by value

Passing by reference

```
> ./test1
a = 2 b= 9
```

Arrays in C

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

void foo(double* A,double B){
        A[0] = 2.0;
        B = 4.0;
}

int main(void){
        double *a;
        int i;
        a = (double*)malloc(sizeof(double)*4);
        for(i=0;i<4;i++){a[i] = (double)i;}
        for(i=0;i<4;i++){printf("a[%d] before=%10.2f\n",i,a[i]);}
        printf("\n");
        foo(a,a[3]);
        for(i=0;i<4;i++){printf("a[%d] after=%10.2f\n",i,a[i]);}
        return 0;
}
```

Arrays in C

```
> ./test3
a[0] before =       0.00
a[1] before =       1.00
a[2] before =       2.00
a[3] before =       3.00

a[0] after =       2.00
a[1] after =       1.00
a[2] after =       2.00
a[3] after =       3.00
```

Viewing what is in an object or executable file

```
> nm test3
0000000000600e40 d _DYNAMIC
0000000000600fe8 d _GLOBAL_OFFSET_TABLE_
00000000004007b8 R _IO_stdin_used
                 w _Jv_RegisterClasses
0000000000600e20 d __CTOR_END__
0000000000600e18 d __CTOR_LIST__
0000000000600e30 D __DTOR_END__
0000000000600e28 d __DTOR_LIST__
00000000004008b0 r __FRAME_END__
0000000000600e38 d __JCR_END__
0000000000600e38 d __JCR_LIST__
0000000000601030 A __bss_start
0000000000601020 D __data_start
0000000000400770 t __do_global_ctors_aux
0000000000400530 t __do_global_dtors_aux
0000000000601028 D __dso_handle
                 w __gmon_start__
0000000000600e14 d __init_array_end
0000000000600e14 d __init_array_start
00000000004006d0 T __libc_csu_fini
00000000004006e0 T __libc_csu_init
                 U __libc_start_main@@GLIBC_2.2.5
0000000000601030 A _edata
0000000000601040 A _end
00000000004007a8 T _fini
0000000000400470 T _init
00000000004004e0 T _start
000000000040050c t call_gmon_start
0000000000601030 b completed.7382
0000000000601020 W data_start
0000000000601038 b dtor_idx.7384
```

Some other difference between C/C++

- In C, all variables have to be declared at the beginning of a function, C++ can have variables declared everywhere.
- C provides some modest OO programming capabilities through the `struct` data structure.
- Function overloading is not valid in C.
- The gap between C and C++ performance is not as wide as in past.

A word about C in Python

- Cython – tries to make up for the poor performance of Python by allowing you to directly import C functions as modules in Python (f2py is the Fortran equivalent that comes with NumPy)

```
➢ cython myPython.pyx
➢ gcc -c -fPIC -O3 -I/usr/include/python2.7 myPython.c
➢ gcc -shared myPython.c -o myPython.so

➢ ...
➢ In Python
  import myPython

➢ a = python.foo(var1,var2)
```

Hands-on Cython and f2py

- Complete the Cython tutorial at http://blog.perrygeo.net/2008/04/19/a-quick-cython-introduction/
  - Note, there are some issues with spacing things like “< =“ in the c code that are placed there on purpose, so you can’t only copy and paste everything.
- Try the f2py from NumPy to do the same thing.
  - Fortran code for the Great Circle is available at http://www.psc.edu/~stbrown/ftest.f90.
  - To make a python module:
    - f2py –c –m <moduleName> <fortranSourceName>
  - Try this two ways, call great_circle with looping in Python, and then call great_circle_loop so that it is all done in Fortran.