程式設計
Computer Programming

Tsung-Chu Huang

Department of Electronic Engineering
National Changhua University of Education

Fall, 2004

Advantages of Using Multiple Files

- Teams of programmers can work on programs. Each programmer works on a different file.
- An object oriented style can be used. Each file defines a particular type of object as a datatype and operations on that object as functions. The implementation of the object can be kept private from the rest of the program. This makes for well structured programs which are easy to maintain.
- Files can contain all functions from a related group. For Example all matrix operations. These can then be accessed like a function library.
- Well implemented objects or function definitions can be re-used in other programs, reducing development time.
- In very large programs each major function can occupy a file to itself. Any lower level functions used to implement them can be kept in the same file. Then programmers who call the major function need not be distracted by all the lower level work.
- When changes are made to a file, only that file need be re-compiled to rebuild the program. The UNIX make facility is very useful for rebuilding multi-file programs in this way.

Compiling Multi-File System

- This process is rather more involved than compiling a single file program.
- Example:

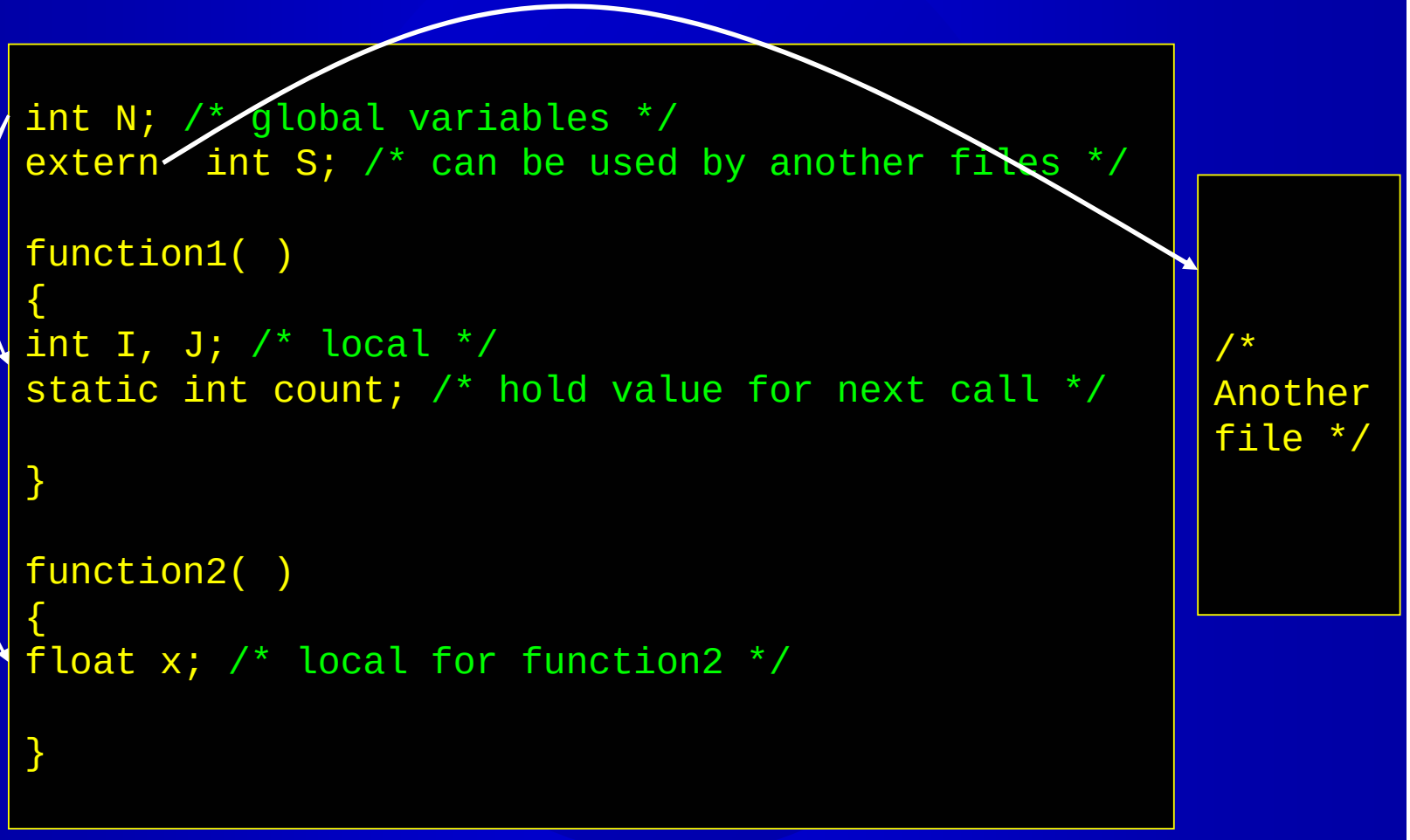
```
cc prog.c func1.c func2.c -o prog
```

- Systematic Management of Complied Files
  - make: usually used in a UNIX system
  - Build: usually used in a Windows system with GUI
  - Example of an Makefile:

```
Main = Vtree
Flag = -g
Obj = MyStr.o Parser.o Tree.o
Main:          $(Obj) $(Main).c
               gcc $(Flag) -o $(Main) $(Obj) $(Main).c
MyStr.o:       MyStr.c
               gcc -c MyStr.c
Parser.o:      Parser.c MyStr.o
               gcc $(Flag) -c Parser.c
Tree.o:        Tree.c Tree.h
               gcc -c Tree.c
```

Life and Range of Variables

- (Local ←→ Static)←→ (Global←→Extern)



```
int N; /* global variables */
extern  int S; /* can be used by another files */

function1( )
{
int I, J; /* local */
static int count; /* hold value for next call */

}

function2( )
{
float x; /* local for function2 */

}
```

```
/*
Another
file */
```

Call by Reference or Value

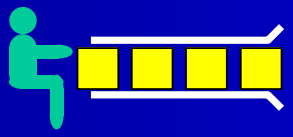
- Call by Value

```
main( )
{
:
      my_printf(“%d”, N);
:
}


My_printf(Format, X)
char Format[];
int X;
{
:
Process(X);
:
}
```

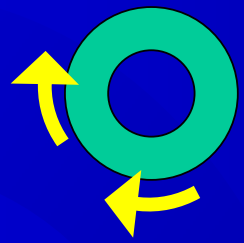
- Call by Reference

```
main( )
{
:
      my_scanf(“%d”, &N);
:
}


My_scanf(Format, X)
char Format[];
int X;
{
:
      X* = some_value;
:
}
```

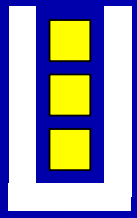
Brief Introduction to Basic Data Structure



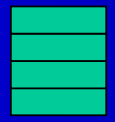
Queue



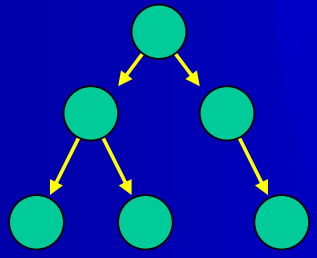
Array with modulo index



Stack

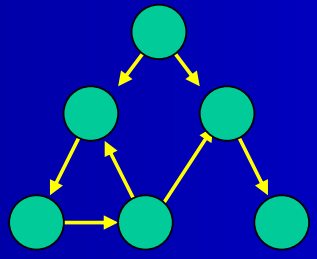


Array with bounded index



Tree

Linked struct



Graph

Exercises

- Exercises of multiple files using extern variables and public functions.
- Exercises of pointer usage in binary trees.