Announcements

- Program #1A
  - due one week from today (next Thursday)
- Submit
  - be careful of spelling of things like “Invalid” and “Instruction”
  - for this project they are all public tests
- Environment
  - must use Linux
- Reading
  - Should have already read Chapter 1-4
  - Chapter 7 [skip 7.6] (by next class)
- Email
  - send mail from a umd account only
  - spam filters discarding aol, hotmail, etc. e-mails

Make: A Tool to Compile Programs

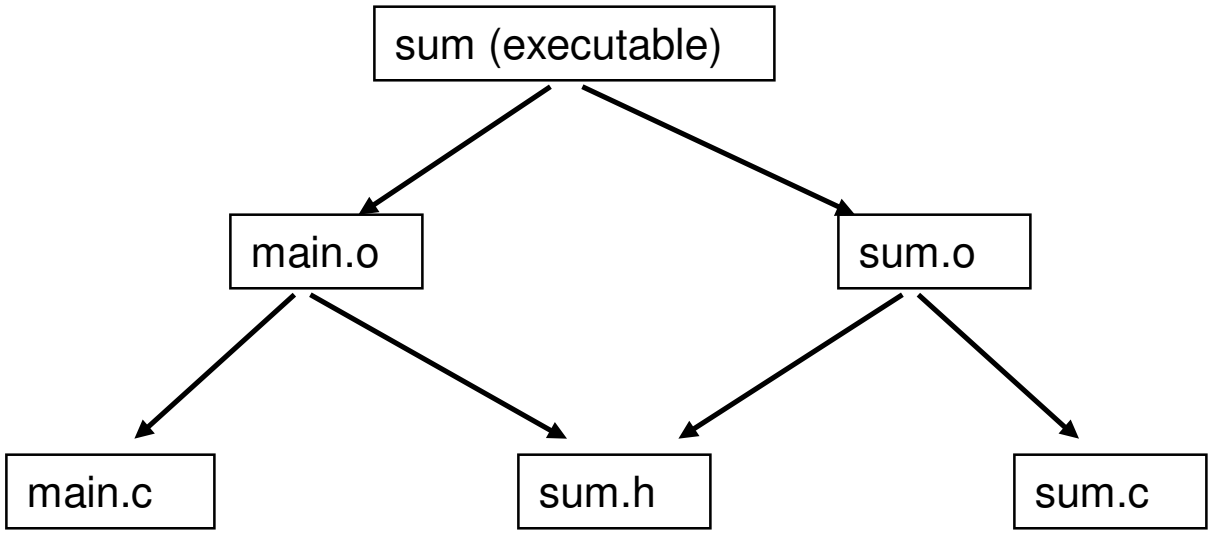
- Problem: Compiling programs is often tedious, accident prone and wasteful of time
- Solution: Automate it!

- Done in Unix by the make command
  - Uses a file called Makefile
    - A makefile is a file (script) containing :
      - Project structure (files, dependencies)
      - Instructions for file creation
- Make is not limited to C programs

Adapted from: www.cs.tau.ac.il/~dcor/Software1/Makefile.**ppt**

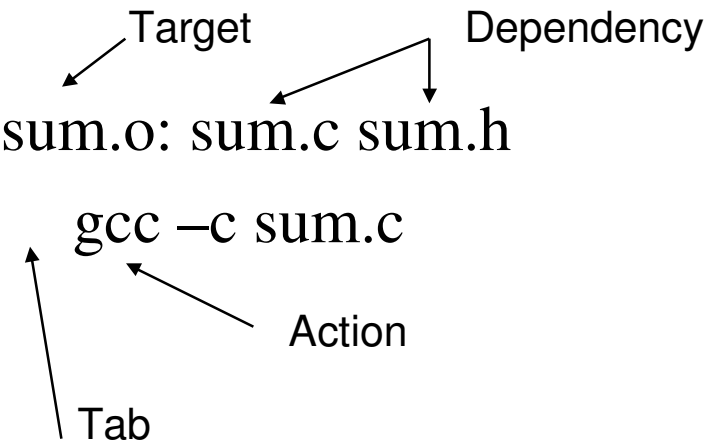
Describing Files to Make

- Makefile contains project dependencies
  - represented as a DAG (= Directed Acyclic Graph)
- Example:
  - Program contains 3 files
  - main.c., sum.c, sum.h
  - sum.h included in both .c files
  - Executable should be the file named sum



Makefile for Previous Example

```
sum: main.o sum.o
    gcc –o sum main.o sum.o

main.o: main.c sum.h
    gcc –c main.c

sum.o: sum.c sum.h
    gcc –c sum.c
```



- Notes About Makefiles:
  - Target is first item in a dependency
  - Items after ":" required to build target
  - Actions must have a tab at start
  - First target in file is the default root

Make's Operation

- Project dependencies tree is constructed
- A target is out of date when any one of its dependencies (directly or indirectly) is newer than itself.
- If out of date, recreate the target
  - Perform action specified
  - Done on the way up the tree
- touch
  - UNIX command that allows you to change the modification time
  - don't use this unless you are sure (can prevent necessary compilations)
- Notes:
  - make ensures minimum compilation, with proper project structure
  - Avoid writing something like:

```
prog: main.c sum1.c sum2.c sum.h
    gcc –o prog main.c sum1.c sum2.c
```

    - this requires full compilation when something changes

Additional Make Features

- Macros
  - <variable> = <value>
  - Example:

```
CC          =   gcc
CFLAGS    =   -g -Wall
Foo.o:   foo.c   foo.h
        $(CC) $(CFLAGS) –c foo.c
```

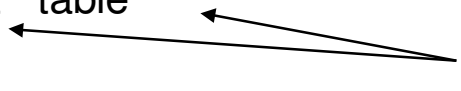
```
Foo.o:   foo.c   foo.h
        gcc   -g –Wall  –c foo.c
```

- Default or Implied Action
  - $(CC) –c $(CFLAGS)
- Default or Implied Dependencies
  - xxx.o must be created from a source file named xxx.?
  - in our case xxx.o comes from xxx.c
- Actions can sometimes delete files not create them
  - Example:

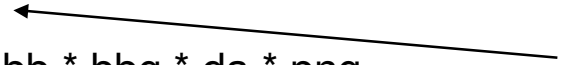
```
clean:
        rm *.o
```

More Complex Makefile

```
# Created for Class Example

all:    intCount   table

clean:
        rm -f *.o *.bb *.bbg *.da *.png

intCount:       intCount.o
        $(CC) -o intCount intCount.o

table:  table.o main.o hashTable.o
        $(CC) -o table table.o main.o hashTable.o

table.o: table.c table.h
hashTable.o:  hashTable.c table.h
main.o: main.c table.h
intCount.o: table.h

CC = gcc
CFLAGS = -g -Wall
```

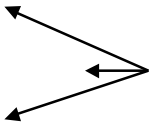


Two top-level programs

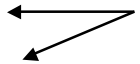


Not part of main tree
built by typing “make clean”

Links together several files



Uses Default Rule



Macro Definitions

Data Types in C

- Integer Family

name	minimum by ANSI standard	ours
– char	8 bits	8 bits
– short	16 bits	16 bits
– int	16 bits	32 bits
– long	32 bits	32 bits
– long long	64 bits	64 bits

- Floating Point

– float	32 bits	32 bits
– double	64 bits	64 bits
– long double	80 bits	96 bits

- Signed by default, but all can be made unsigned
  - integer types are defined in the ANSI standard: unsigned int or unsigned long
  - unsigned floating types are not in the ANSI standard but are on some compilers
- Literals (examples)
  - Decimal 255
  - negative signed decimal int -255
  - Hex 0xff
  - floating point types: 3.14159, 1E10, 25., 6.023e23
  - Characters: ‘a’, ‘\n’
  - C-Strings: “a long dull string”, “\n”, “”

Data Types in C (cont.)

- Pointers
  - holds an address
  - points to a specific storage space
  - usually declared to point to a space of a certain type
  - space pointed at can be dynamically allocated (later) but this is not necessary

```
#include <stdio.h>

int main(){
  double a;
  double *b;

  printf("size of a is %d and size of b = %d\n",sizeof(a), sizeof(b));

  a = 5;
  b = &a;
  printf("a = %.2f and b indicates %.2f\n",a,*b);

  a = 21;
  printf("a = %.2f and b indicates %.2f\n",a,*b);

  return 0;
}
```

Data Types in C (cont.)

- Enumerated Types
  - Declaring an enumerated type:
    - enum opType { plusOperator, minusOperator };
  - Declaring a variable of an enumerated type:
    - enum opType opName;
  - Using an enumerated type in an expression:
    - opName = plusOperator;
  - Can control values of enumerated type:
    - enum opType { plusOperator=3, minusOperator =8};

```
#include <stdio.h>
int main(void){
  enum opType {plusOperator, minusOperator};
  enum opType opName1, opName2;
  opName1 = minusOperator;   opName2 = plusOperator;
  if (opName1 < opName2)
        printf("first is less\n");
   else
        printf("second is less\n");
  printf("first is %d and second is %d\n",opName1, opName2);
  return 0;
}
```

Variable Declarations

- <type> <variable1>, <variable2>, … ;
  - int a, b, c;
  - short d; or short int d;
- Variables names
  - May not be reserved words
  - May contain alphanumeric characters and underscore
    - (A-Z, a-z, 0-9, and "_")
    - examples:
      - aValid_Variable_Name
      - aValid_Variable_Name_2
      - a2
  - Must start with non-numeric: A-Z,a-z or _
- May initialize variables as part of declaration
  - int a = 4;
- Implicit Declarations (Book section 3.2.4)
  - Skip section book and don't ever use

Simple Arrays

- Array dimensions must be known at compile time
- Array Declaration
  - int a[10]; /* array of 10 integers */
  - really one big chuck of space – big enough for 10 integers
- Accessing array elements
  - Arrays start from 0 and go to n-1 elements
  - Compiler will let programmer access invalid elements
    - Can lead to many problems that are hard to find
    - mathematically calculates the “offset”
  - Examples of array references (assume: int i = 2);
    - a[0]
    - a[i]
    - a[i+3]
      - what if int i = 7; ??

Typedef

- Sometimes you want to name your own type
  - Makes program easier to read
  - Makes it easier to change types later
- typedef double dollars;

Variable Scope

- File Scope
  - Entire file
  - Not declared inside any function
- Block Scope
  - Before any executable statement in a block of code { .. }
- Repeating same variables in nested scope is allowed but should be avoided!!

```
main() {
    int a;
    { int a; a = 3; }
}
```

Storage and Linkage

- Linkage
  - Determines how multi-file visibility issues are resolved
  - extern - There is only one instance of this variable
  - static - This variable is private to this file

    foo.c:

    ```
    static int a;
    ```

    bar.c:

    ```
    static int a;
    ```
- Storage Classes
  - Default is automatic:
    - one variable instance per function instance
    - lifetime limited to while function is active
      - running or in called subroutine
  - Static changes default
    - one variable instance
    - lifetime is the current run of the entire program

Storage Class and Linkage Example

```
#include <stdio.h>
#include "funct.h"

extern int a;
static int b = 3;

int main(void){
  printf("%d and %d\n", a,b);
  funct1();
  printf("%d and %d\n", a,b);
  funct1();
  printf("%d and %d\n", a,b);
  return 0;
}
```

```
#include <stdio.h>
#include "funct.h"

int a = 33;
static int b = 99;

void funct1(void){
  int m = 3;
  static int n = 5;
  printf("at beginning of function a = %d and b = %d\n",a,b);
  printf("and m = %d and n = %d\n",m,n);
  a = 27, b = 20;
  m += 100; n += 200;
  printf("at beginning of function a = %d and b = %d\n",a,b);
  printf("and m = %d and n = %d\n",m,n);
  return;
}
```

```
33 and 3
at beginning of function a = 33 and b = 99
and m = 3 and n = 5
at beginning of function a = 27 and b = 20
and m = 103 and n = 205
27 and 3
at beginning of function a = 27 and b = 20
and m = 3 and n = 205
at beginning of function a = 27 and b = 20
and m = 103 and n = 405
27 and 3
```