CS 2213
Advanced Programming

Ch 3 – Overview – C programming Language

Interfaces – Libraries

String, I/O, Math, Char Libraries

Turgay Korkmaz

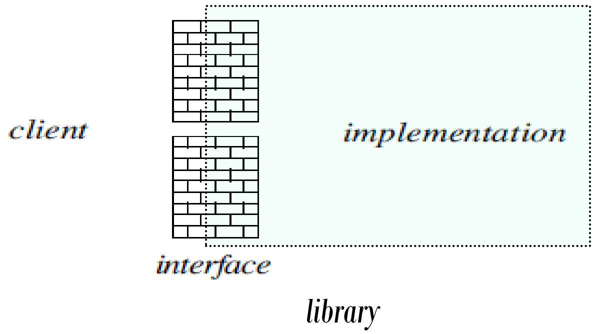
Office: SB 4.01.13
Phone: (210) 458-7346
Fax: (210) 458-4437
e-mail: korkmaz@cs.utsa.edu
web: www.cs.utsa.edu/~korkmaz

Objectives

- To appreciate the importance of **interfaces** and **libraries**
- To understand the **terminology** used in interface-based programming.
- To recognize the criteria used to evaluate the design of an interface.
- To learn the syntactic rules and conventions required to write an interface file.

- To design an example interface/library, namely **random.h**
  - To be able to use the facilities provided by the **random.h** interface**.**
- To understand how strings are represented
- To learn how to use the standard C **string.h** and textbook's **strlib.h**
- To learn how to use the standard C **stdio.h** to read and write data files
- To understand other standard libraries (**math.h**, **ctype.h** etc.)

Introduction to



- Programmers depend on **libraries**
- There is a distinction between the library itself and other programs called its **clients** (application or driver programs) that make use of libraries.
- The boundary between a library and its clients is called the **interface**
  - Provides a channel of communication
  - Acts as a barrier that prevents complex details on one side from affecting the other (ABSTRACTION)

You may have two hats

Library Developer

needs to know both

what a library does

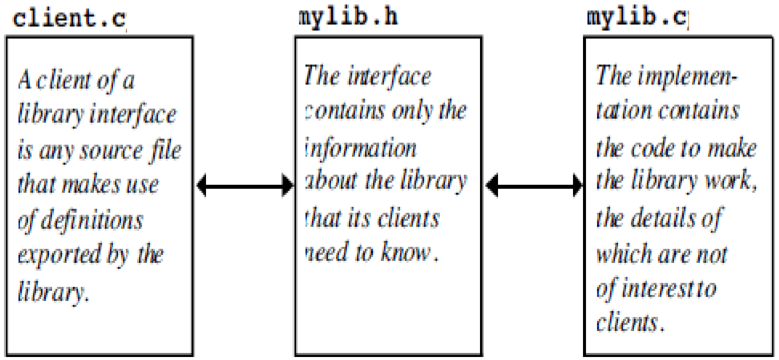
and

how it does

Application Developer

just needs to know
what a library does,
but he/she does not care
how it does

Interfaces and Implementations



client.c

A client of a library interface is any source file that makes use of definitions exported by the library.

mylib.h

The interface contains only the information about the library that its clients need to know.

mylib.c

The implementation contains the code to make the library work, the details of which are not of interest to clients.

- Suppose we want to develop several functions and make them available to clients as a library, then we need to have two files:
  - An interface file called **header file** `mylib.h`
    - Contains function prototypes
    - Export data types and constants
  - An implementation file `mylib.c`
    - Contains actual implementation of the functions
- Clients can now use `mylib` library

Package and abstraction

Standard vs. User defined libraries

- We already used several standard libraries and the ones provided by the textbook

```
#include <stdio.h>
#include “genlib.h”
```

- We will now learn how to design and implement new libraries and use them in our driver/application programs

Principles of good interface/library design

- **Unified.** A single interface should define a consistent abstraction with a clear unifying theme. If a function does not fit within that theme, it should be defined in a separate interface.
- **Simple.** To the extent that the underlying implementation is itself complex, the interface must hide as much of that complexity from the client as possible.
- **Sufficient.** When clients use an abstraction, the interface must provide sufficient functionality to meet their needs. If some critical operation is missing from an interface, clients may decide to abandon it and develop their own, more powerful abstraction. As important as simplicity is, the designer must avoid simplifying an interface to the point that it becomes useless.

"Everything should be made as **simple** as possible, but not simpler, " Albert **Einstein**

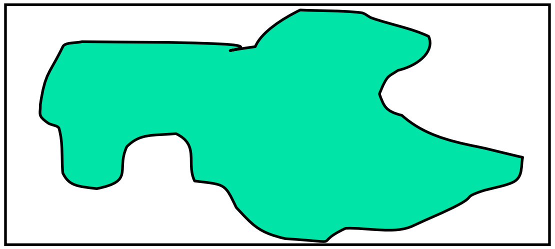
- **General.** A well-designed interface should be flexible enough to meet the needs of many different clients. An interface that performs a narrowly defined set of operations for one client is not as useful as one that can be used in many different situations.
- **Stable.** The functions defined in an interface should continue to have precisely the same structure and effect, even if their underlying implementation changes. Making changes in the behavior of an interface forces clients to change their programs, which compromises the value of the interface.

A simple library example: Random numbers

- What is a random number?
- Do standard C libraries provide any help?
- Design and implement a random number library…

What is a Random Number?

- Tossing a coin (0, 1) Rolling a die (1, 2,…6)
- Min, Max, Avg, possible outcomes are equally likely or not,
- Many problems require use of random numbers, here is an example
  - How can you compute the area of an irregular shape?
  - Simulations



Uniform Random numbers

- All outcomes are equally likely
- For example fair die, where each outcome has the same probability of 1/6,
- So we can generate uniform random numbers between 1 and 6 by rolling a die.
- What if we need random numbers in another range? For example, 1 and 100?

Uniform Random numbers (cont’d)

- Standard C library `stdlib.h` has `rand()`
  - generates random numbers between 0 and RAND_MAX
  - RAND_MAX is a system dependent constant (e.g., 32,767) defined in `stdlib.h`
- What will be the output when we execute

```
#include <stdlib.h>
main()
{
   printf(“%d %d %d\n”,rand(), rand(), rand());
}
```

- What will be the output, if we re-run the same program?

Pseudo-random Numbers

- Computers generate random numbers using a **seed** number and an **algorithm**.
- So, if you give the same seed,
  - you will always get the same sequence of numbers *called **pseudo-random** numbers*
- Standard C library `stdlib.h` has `srand(int seed)`
  - allows us to give a new seed number

Example: generate 10 RNs

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

int main(void)
{
   /*  Declare variables.  */
   int seed;
   int k;

   /*  Get seed value from the user.  */
   printf("Enter a positive integer seed value: \n");
   scanf("%d", &seed);
   srand(seed);

   /*  Generate and print ten random numbers.  */
   printf("Random Numbers: \n");
   for (k=1; k<=10; k++)
      printf("%i ", rand());
   printf("\n");

   /*  Exit program.  */
   return 0;
}
```

`rand()` and `srand()` are not enough…

- What if we want to get
  - random numbers in the range [200 500]?
  - real random numbers in the range [0.5 1.0]
  - random numbers from other distributions (e.g., exponential, normal etc.)
- We can develop a new “Random Number” library providing all these functions while hiding their implementation details from client programs

The structure of the random.h interface

```
/* Comments are removed here. Please see the textbook */
#ifndef _random_h
#define _random_h

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "genlib.h“

int RandomInteger(int low, int high);

double RandomReal(double low, double high);

bool RandomChance(double p);          /* RandomChance(.30) returns
                                       TRUE 30 percent of the time. */

void Randomize(void);                /* This function initializes the random-
                                      number generator based on time */

#endif
```

```
#ifndef _random_h
#define _random_h
   /* … */
#endif
```

What is the purpose of these boilerplate lines?

The purpose of the interface boilerplate is to prevent the compiler from reading the same interface many times during a single compilation.

Implementation of the random.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "genlib.h"
#include "random.h"

int RandomInteger(int low, int high)
{
   int k;
   k = rand() %
        (high - low + 1);

   return (low + k);
}
```

```
/* Comments are removed. Please see the textbook */

/* This library uses primitive random number generation
functions provided by standard C library… */
```

```
double RandomReal(double low, double high)
{
    double d;
    d = (double) rand() /
        ((double) RAND_MAX + 1);
    return (low + d * (high - low));
}
bool RandomChance(double p)
{
    return (RandomReal(0, 1) < p);
}
void Randomize(void)
{
    srand((int) time(NULL));
}
```

Constructing a client program: HiLo Game

```
/* Write a program that allows
a user to play HiLo game. User
wins if he/she can guess the
number between 1-100 within at
most 6 iterations  */
```

```
#include <stdio.h>
#include "genlib.h"
#include "random.h"

void playHiLo( int s);          /* prototype */

int main(void)
{
   int secret;                  /*  Declare variables */

   Randomize();

   while(1){
     secret = RandomInteger(1,100);
     playHiLo(secret);
   }
   return 0;
}
```

Client: HiLo Game (cont’d)

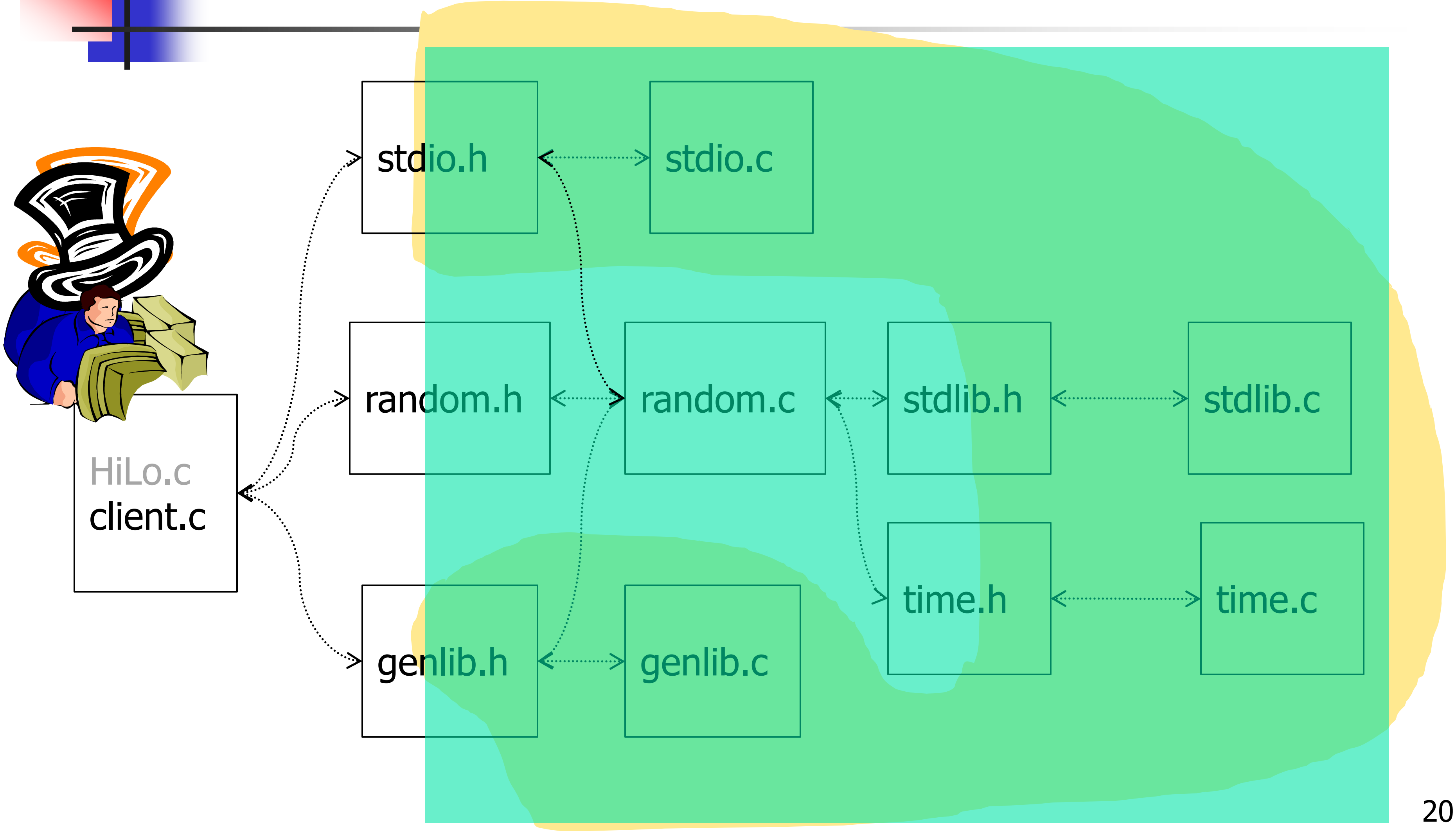
```
void playHiLo(int s)
{
  int i, guess;

  for(i=1; i <=6; i++){
    printf("Enter your guess : ");
    scanf("%d", &guess);
    if (guess > s)
       printf("It is Higher than secret\n");
    else if (guess < s)
       printf("It is Lower than secret\n");
    else {
         printf("Cong! you won\n");
         return;
    }
  }
  printf("Sorry! Try again\n");
  return;
}
```

Exercise: Another "guess the number game"

- Computer selects a random number *s* between [1000 9999]
- User tries to guess it by entering *g*
- Computer tells how many digits are in place, out of place, not in secret number
- For example, if *s* is 6234
  - User enters *g* as 7436, then computer says
    - 1 digit is in place
    - 2 digits are out of place
    - 1 digit is not in secret number
- User keeps trying until he finds the secret number
- How about developing a program where the user selects the random number and computer tries to find it???

Interactions between libraries



Compile

```
> ls
client.c random.c random.h
> gccx client.c random.c -o client
```

- OR you can first compile them individually

```
> gccx -c random.c          WHY?
> gccx -c client.c
> ls
client.c client.o random.c random.h random.o
> gccx client.o random.o -o client
```

- It would be better to use **Makefile** and **make**

Makefile (text file) and make (program)

```
# Makefile comments……

all: client
tidy:
   rm -f ,* .,* *~ core a.out *.o graphics.ps

# C compilations
client.o: client.c random.h
   gcc -c client.c
random.o: random.c random.h
   gcc -c random.c

# Executable programs
client: client.o random.o
   gcc -o client client.o random.o
```

What if you want to use **gccx** instead gcc

```
> ls
client.c Makefile
random.c random.h

> make client

> ls

> make tidy

> ls

> make

> ls
```

http://en.wikipedia.org/wiki/Make_(software)

Makefile (text file) and make (program)

```
# Makefile comments……
PROGRAMS = client
CC = gcc
CFLAGS =
all: $(PROGRAMS)
tidy:
   rm -f ,* .,* *~ core a.out *.o graphics.ps
# C compilations
client.o: client.c random.h
   $(CC) $(CFLAGS) -c client.c
random.o: random.c random.h
   $(CC) $(CFLAGS) -c random.c
# Executable programs
client: client.o random.o
   $(CC) $(CFLAGS) -o client client.o random.o
```

To use books library, put
CC = gccx
instead of CC = gcc

```
> ls
client.c Makefile
random.c random.h

> make client

> ls

> make tidy

> ls

> make

> ls
```

http://en.wikipedia.org/wiki/Make_(software)

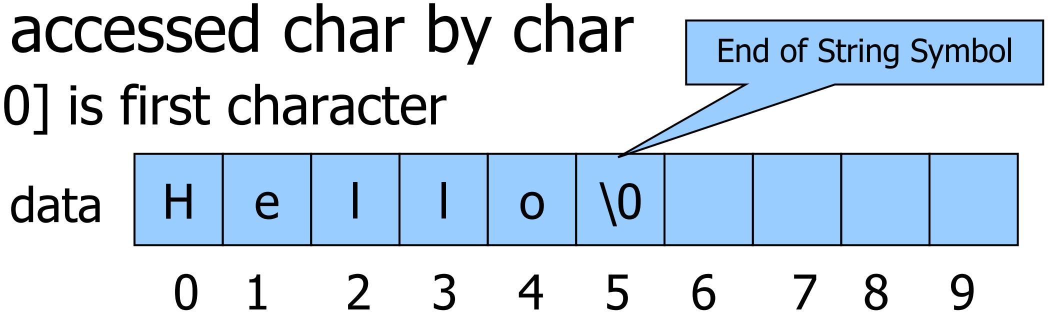
Strings: Arrays of Characters

- In Ch2, we studied strings as arrays of characters terminated by null
- There is a standard `string.h` library
- The textbook defines string as a new type in `genlib.h` and develops a `strlib.h` library on top of the standard string library

Representation of Strings

Also discussed before in ch2 as Array of Characters

- A string is an *array* of characters
  - `char data[10] = “Hello”; // data is a constant !`
  - `char data2[] = {‘H’, ‘e’, ‘l’, ‘l’, ‘o’, ‘\0’}`
  - `char *data3 = “Hello”; // “Hello” is a constant !`
  - `string data4 = “Hello”; /* if we use genlib.h */`
- Use printf to print strings
  - `printf(“%s”,data); scanf(“%s”,data);`
  - `sprintf(data, “%d ”, X); sscanf(data3, “%d ”, &X);`
  - `data3 = GetLine(); /* Textbook’s lib*/`
- Can be accessed char by char
  - data[0] is first character



End of String Symbol

data

H	e	l	l	o	\0				
0	1	2	3	4	5	6	7	8	9

Strings: arrays vs. pointers

```c
/* Array implementation */

static int CountSpaces(char str[])
{
  int i, nSpaces;


  nSpaces = 0;
  for (i = 0; str[i] != '\0'; i++)
  {
     if (str[i] == ' ') nSpaces++;
  }
  return (nSpaces);
}
```

```c
/* Pointer implementation */

static int CountSpaces(char *str)
{
 int nSpaces;
 char *cp;

 nSpaces = 0;
 for (cp = str; *cp != '\0'; cp++)
 {
    if (*cp == ' ') nSpaces++;
 }
   return (nSpaces);
}
```

`string` type in genlib.h and strlib.h and string.h libraries

- In genlib.h, the texbook defines

  `typedef char *string;`

- So `string` is identical to `char *`
- The biggest difference between `strlib.h` and `string.h` is Memory allocation
  - `<string.h>` functions assumes that user/client allocates memory for the characters in string
  - `“strlib.h”` functions dynamically allocate memory for the characters in string

Common Functions exported by standard `string.h`

```
size_t strlen(const char *str);

char *strcpy(char *dest, const char *src);

char *strncpy(char *dest, const char *src, size_t n);

char *strcat(char *dest, const char *src);

char *strncat(char *dest, const char *src, size_t n);

int strcmp(const char *str1, const char *str2);

int strncmp(const char *str1, const char *str2,size_t n);

char *strchr(const char *str, int c);

char *strstr(const char *str1, const char *str2);
```

… more …

Common Functions exported by textbook's `strlib.h`

```
string Concat(string s1, string s2);
char IthChar(string s, int i);
string SubString(string s, int p1, int p2);
string CharToString(char ch);
int StringLength(string s);
string CopyString(string s);
boolean StringEqual(string s1, string s2);
int StringCompare(string s1, string s2);
int FindChar(char ch, string text, int start);
int FindString(string str, string text, int start);
string ConvertToLowerCase(string s);
string ConvertToUpperCase(string s);
string IntegerToString(int n);
int StringToInteger(string s);
string RealToString(double d);
double StringToReal(string s);
```

Some of functions dynamically allocate memory.

But you need to **free** them…
So you need to know the pointers…

See original strlib.h and strlib.c files available at ~korkmaz/cslib and at the class web page

Client program: Convert an English word to PigLatin by applying the following rules

- If the word contains no vowels, no translation is done, which means that the translated word is the same as the original.
- If the word begins with a vowel, the function adds the string **"way"** to the end of the original word. Thus, the Pig Latin equivalent of ***any*** *is* ***anyway***.
- If the word begins with a consonant, the function extracts the string of consonants up to the first vowel, moves that collection of consonants to the end of the word, and adds the string **"ay".** For example, the Pig Latin equivalent of ***trash*** *is* ***ashtray***.

PigLatin using string.h

```
static void PigLatin(char *word, char buffer[], int bufferSize)
{
    char *vp;
    int wordLength;
    vp = FindFirstVowel(word);
    wordLength = strlen(word);
    if (vp == word) {
        wordLength += 3;
    } else if (vp != NULL) {
        wordLength += 2;
    }
    if (wordLength >= bufferSize)
         Error("Buffer overflow");
    if (vp == NULL) {
        strcpy(buffer, word);
    } else if (vp == word) {
        strcpy(buffer, word);
        strcat(buffer, "way");
    } else {
        strcpy(buffer, vp);
        strncat(buffer, word, vp - word);
        strcat(buffer, "ay");
    }
}
```

```
static char *FindFirstVowel(char *word)
{
    char *cp;

    for (cp = word; *cp != '\0'; cp++){
        if (IsVowel(*cp)) return (cp);
    }
    return (NULL);
}
static bool IsVowel(char ch)
{
    switch (ch) {
      case 'A': case 'E': case 'I':
      case 'O': case 'U':
      case 'a': case 'e': case 'i':
      case 'o': case 'u':
        return (TRUE);
      default:
        return (FALSE);
    }
}
```

Home Exercise

PigLatin using strlib.h

```
static string PigLatin(string word)
{
  int vp;
  string head, tail;

  vp = FindFirstVowel(word);
  if (vp == -1) {
      return (word);
  } else if (vp == 0) {
      return (Concat(word, "way"));
  } else {
    head = SubString(word, 0, vp - 1);
    tail = SubString(word, vp, StringLength(word) - 1);
    return (Concat(tail, Concat(head, "ay")));
  }
}
```

```
static int FindFirstVowel(string word)
{
  int i;

  for (i = 0; i < StringLength(word); i++) {
     if (IsVowel(IthChar(word, i))) return (i);
  }
  return (-1);
}

/* isVowel is the same as before */
```

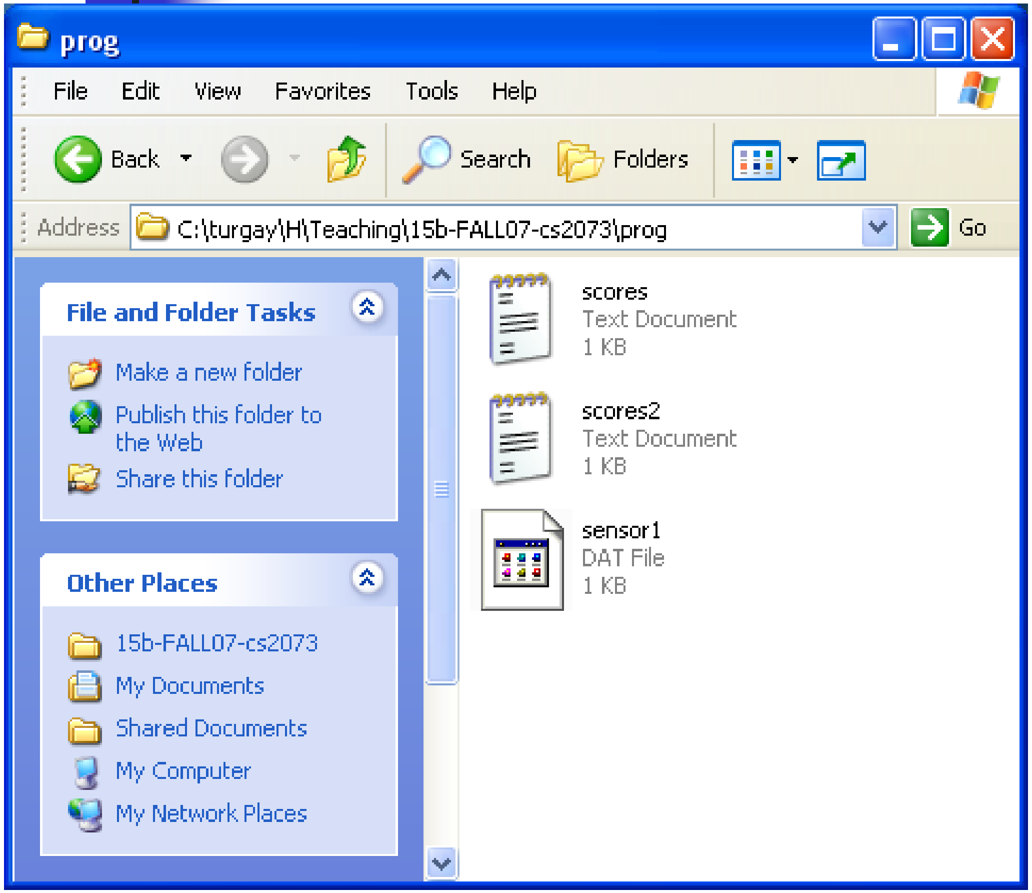
Other libraries

Standard I/O Library

Data Files

- So far, we used
  - **scanf** (also GetInteger, GetReal, GetLine) to enter data
  - **printf** to print data on the screen
- What if
  - we have 1000 data points to enter? Can we still enter them by hand?
  - the output has several lines and we want to store the output results and use them in other programs?

Read Access to Data Files

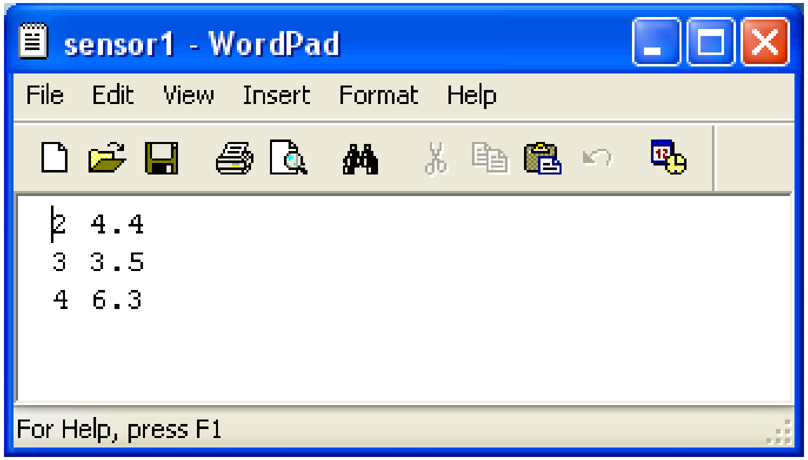


C:\turgay\H\Teaching\15b-FALL07-cs2073\prog

scores
Text Document
1 KB

scores2
Text Document
1 KB

sensor1
DAT File
1 KB



```
2  4.4
3  3.5
4  6.3
```

Read Access to Data Files

- **#include <sdtio.h>**
- File pointer must be defined in C program

FILE *sensor1;

- File pointer must be associated with a specific file using the **fopen** function
- If the program and data file are in the same directory

sensor1 = fopen(“sensor1.dat”, “r”);

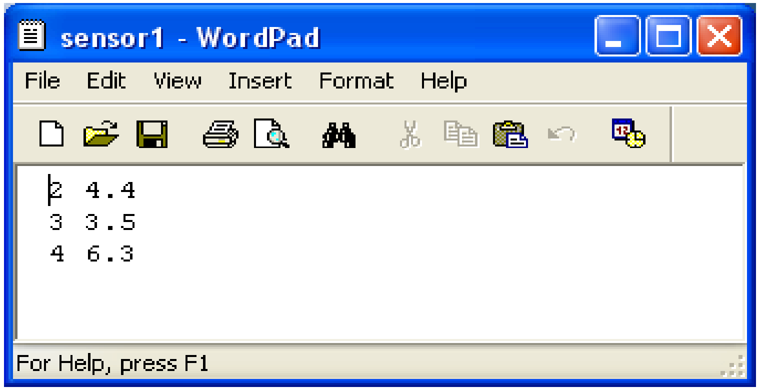
- Else give the full path

sensor1 = fopen(“C:\turgay\H\prog\sensor1.dat”, “r”);

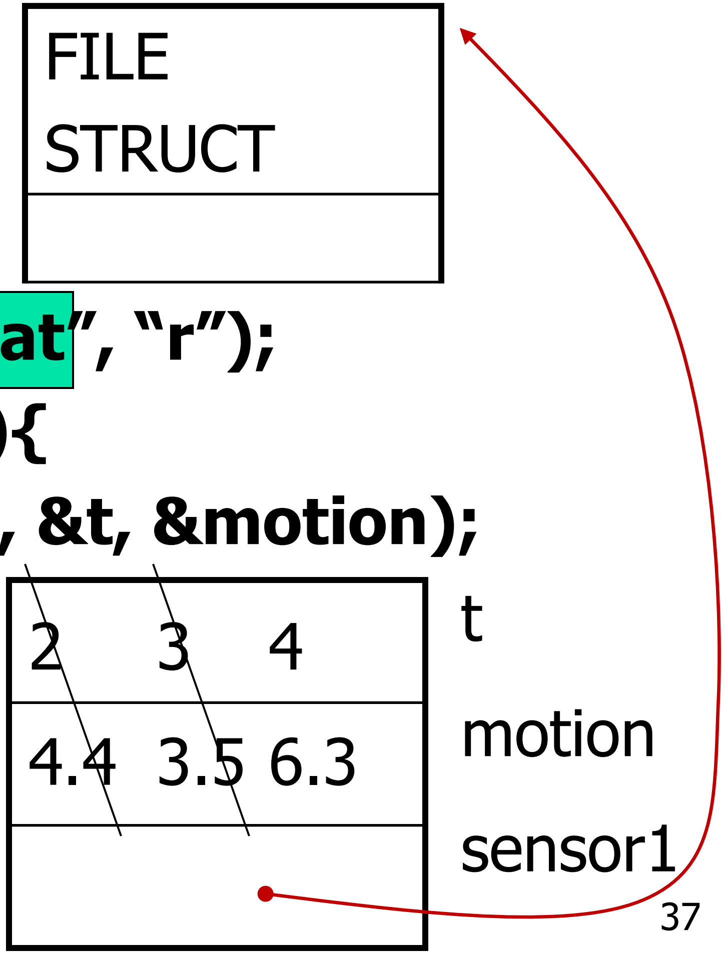
Read from Data Files

- Input file - use fscanf instead of scanf

```
#include <sdtio.h>
FILE *sensor1;
double t, motion;
sensor1 = fopen("sensor1.dat", "r");
while( /* not end of file */ ){
  fscanf(sensor1, "%lf %lf", &t, &motion);
}
```



```
2  4.4
3  3.5
4  6.3
```



FILE
STRUCT

~~2~~ ~~3~~ 4	t
~~4.4~~ ~~3.5~~ 6.3	motion
●	sensor1

Create New Data Files
Write Access to Data Files

- **#include <sdtio.h>**
- File pointer must be defined in C program

  FILE *balloon;
- File pointer must be associated with a specific file using the **fopen** function
- If the program and data file are in the same directory

  balloon = fopen("balloon.dat", "w");
- Else give the full path

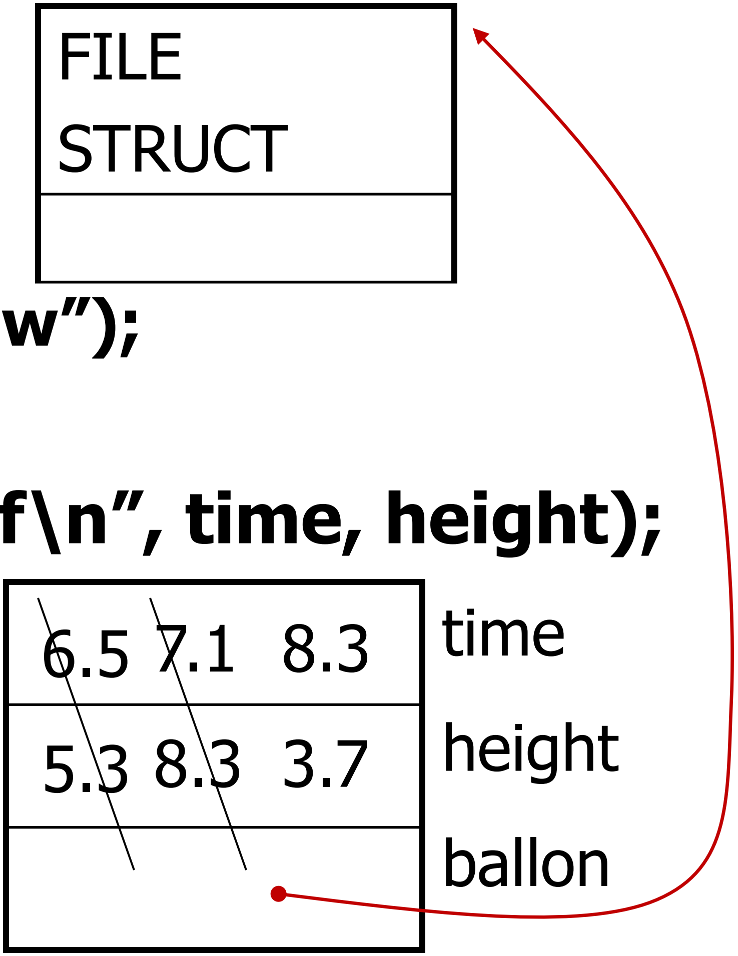
balloon = fopen("C:\turgay\H\Teaching\prog\balloon.dat", "w");

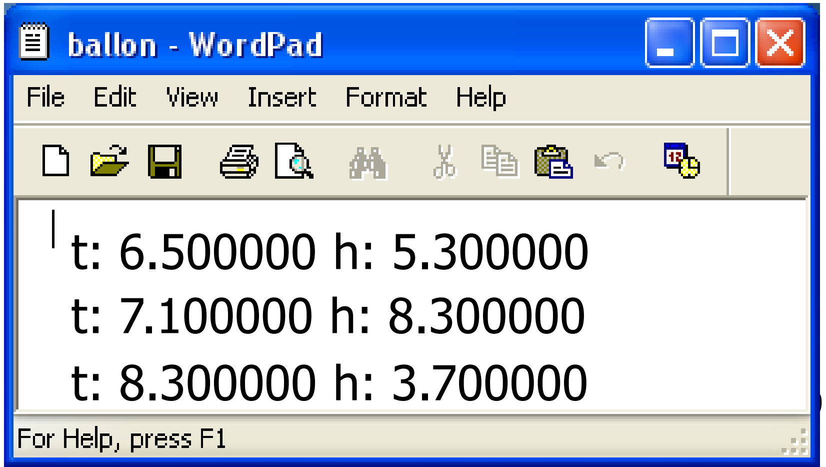
Instead of "w" we can use "a" if we want to file be open for appending

Write to Data Files

Output file - use fprintf instead of printf

```
#include <sdtio.h>
FILE *balloon;
double time=6.5, height=5.3;
balloon = fopen("balloon.dat", "w");
while(/* there is data */)
    fprintf(balloon, "t: %f h: %f\n", time, height);
```



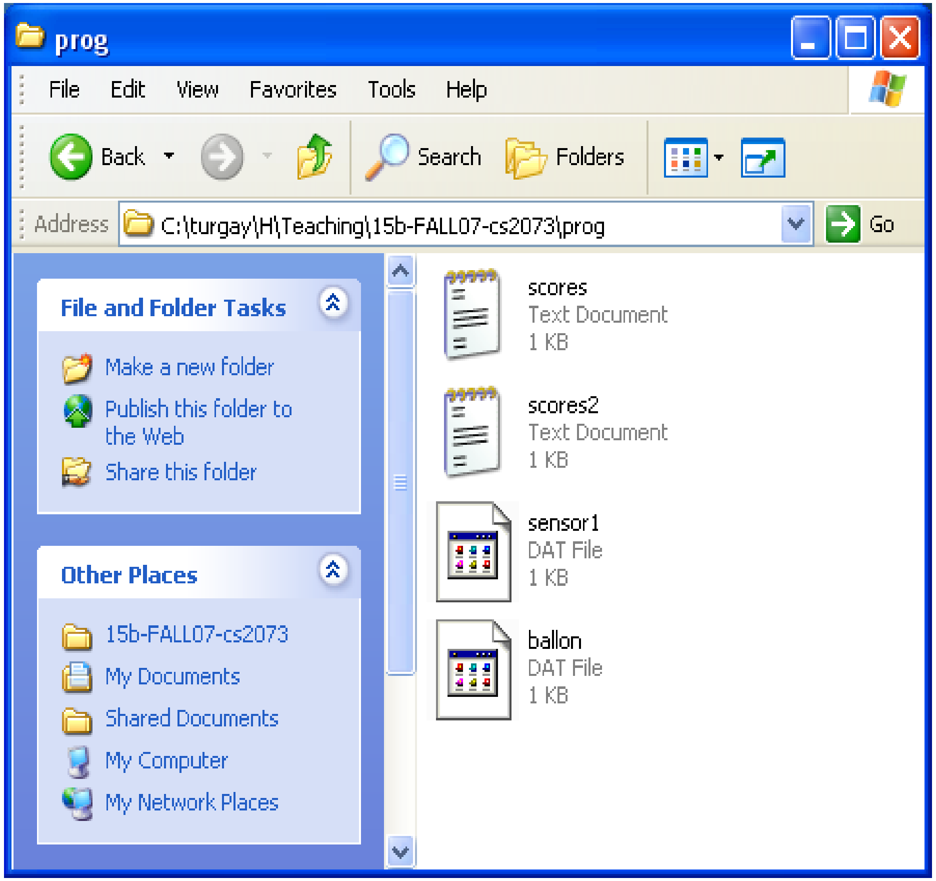


```
t: 6.500000 h: 5.300000
t: 7.100000 h: 8.300000
t: 8.300000 h: 3.700000
```

For Help, press F1

At the end, Use **fclose**

```
fclose(sensor1); fclose(balloon);
```



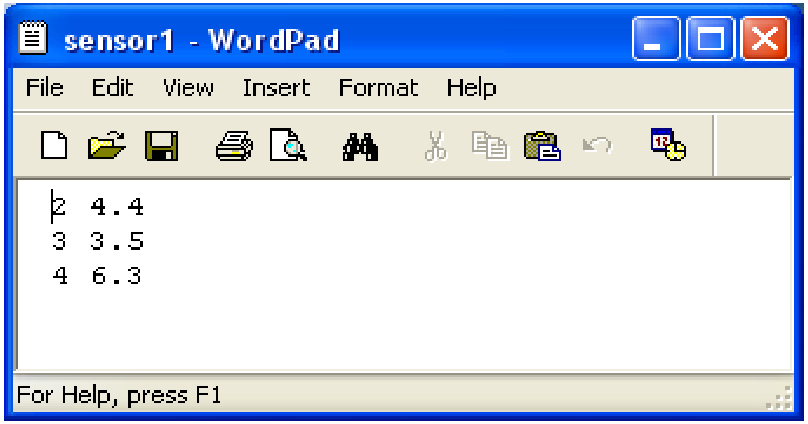
Address: C:\turgay\H\Teaching\15b-FALL07-cs2073\prog

scores
Text Document
1 KB

scores2
Text Document
1 KB

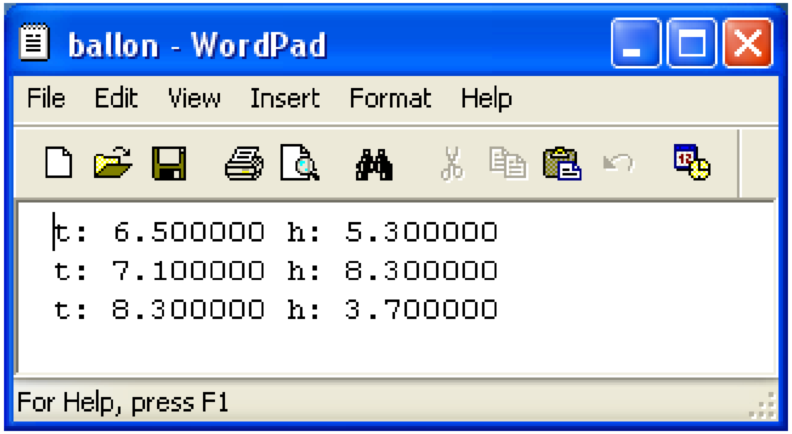
sensor1
DAT File
1 KB

ballon
DAT File
1 KB



```
2  4.4
3  3.5
4  6.3
```

For Help, press F1

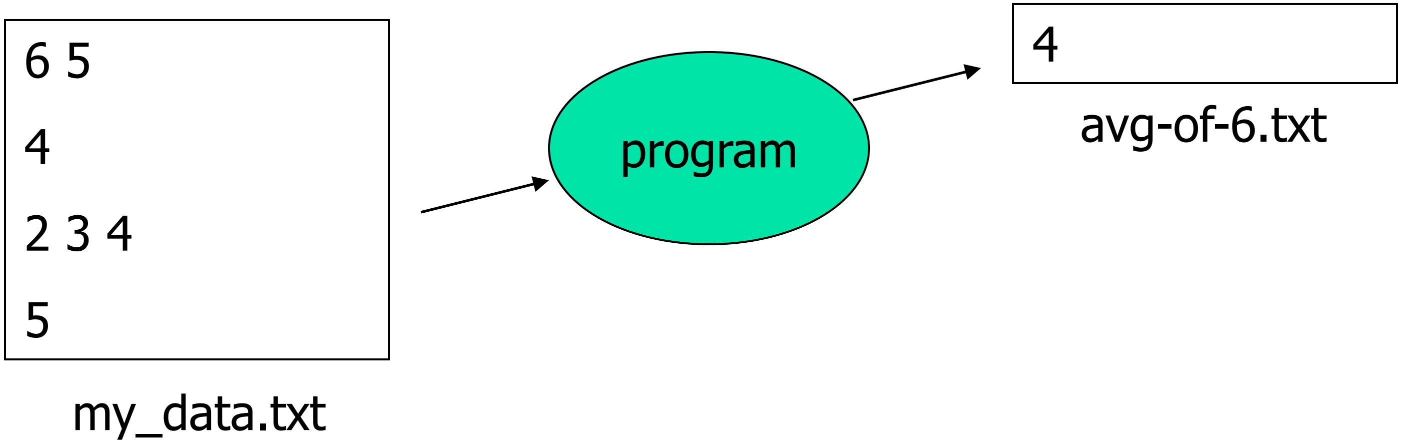


```
t: 6.500000 h: 5.300000
t: 7.100000 h: 8.300000
t: 8.300000 h: 3.700000
```

For Help, press F1

Example

- Read 6 values from a file named my_data.txt and write their average into another file named avg-of-6.txt



```
6 5
4
2 3 4
5
```

my_data.txt

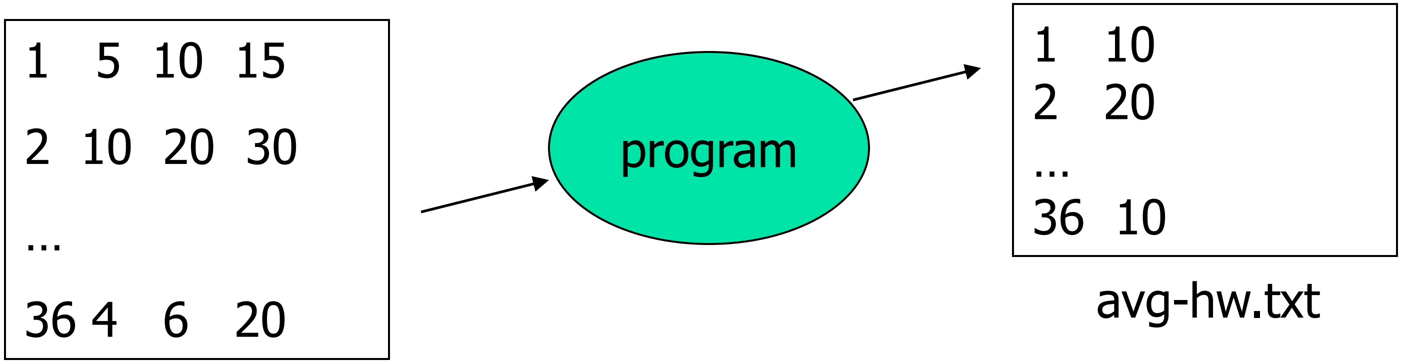
program

```
4
```

avg-of-6.txt

Example: average grade

- Suppose we keep the id and three HW grades of 36 students in a file named grades.txt
- Write a program to compute average grade for each student and write each students avg into another file named avg-hw.txt



```
1   5  10  15

2  10  20  30

…

36 4   6   20
```

grades.txt

```
1   10
2   20
…
36  10
```

avg-hw.txt

Check what fopen, fscanf, fprintf return

```
FILE *fp;
  if ((fp=fopen(“data.txt”, “r”)) == NULL){
   printf(“Program cannot open the file\n”);
   return -1;
}

N=fscanf(fp, “%d %d %d”, &v1, &v2, &v3);
/* N is the number of values read successfully */

while(fscanf(fp, “%d %d %d”, &v1, &v2, &v3) == 3) {
  /* process v1 v2 v3 */
}
```

Reading Data Files

When to stop

- Counter controlled loop
  - First line in file contains count
  - Use `for` loop
- Trailer signal or Sentinel signal
  - Data ends when a special data value is seen -999
  - Use `while` loop
- End of file controlled loop
  - When file is created EOF is inserted
  - Use `while` loop
    - feof(fileptr) == 0 is TRUE if EOF is not reached
    - fscanf cannot read as many values as you wanted when EOF is reached

Counter controlled loop

Usually first line in file contains the count

```
#include <stdio.h>
int main()
{
  FILE *scorefile;
  int score, count, i, sum=0;

  if((scorefile = fopen("scores2.txt","r")) == NULL) ){
    printf(“Program cannot open the file\n”);
    exit(-1);
  }
  fscanf(scorefile,"%d", &count);
  for (i=1; i<=count; i++) {
     fscanf(scorefile,"%d", &score);
     sum = sum + score;
  }
  printf(“Average score %lf \n",(double)sum/count);
  fclose(scorefile);
  return(0);
}
```

```
6
56
78
93
24
85
63
```

scores2.txt

Trailer signal or Sentinel signal

```
#include <stdio.h>
int main()
{
  FILE *scorefile;
  int score, count=0, i, sum=0;

  if((scorefile = fopen("scores3.txt","r")) == NULL) ){
    printf(“Program cannot open the file\n”);
    exit(-1);
  }
  fscanf(scorefile,"%d", &score);
  while(score >= 0) {
    count++;
    sum = sum + score;
    fscanf(scorefile,"%d", &score);
  }
  printf(“Average score %lf \n",(double)sum/count);
  fclose(scorefile);
  return(0);
}
```

```
56
78
93
24
85
63
-999
```

scores3.txt

End of file controlled loop

```
#include <stdio.h>
int main()
{
  FILE *scorefile;
  int score, count=0, i, sum=0;

  if((scorefile = fopen("scores4.txt","r")) == NULL) ){
    printf(“Program cannot open the file\n”);
    exit(-1);
  }
  while (fscanf(scorefile,"%d",&score) == 1) {
    count++;
    sum = sum + score;
  }
  printf(“Average score %lf \n",(double)sum/count);
  fclose(scorefile);
  return(0);
}
```

```
56
78
93
24
85
63
```

scores4.txt

```
while (feof(scorefile) == 0) {
  fscanf(scorefile,"%d",&score);
  count++;
  sum = sum + score;
}
```

Exercise

- In previous three programs, we found average.
- Suppose, we want to also know how many data points are greater than average.
- Change one of the previous programs to determine the number of data points that are greater than average.

Exercise

- Given a file of integers. Write a program that finds the minimum number in another file.

- Algorithm to find minimum in a file:

```
open file
set minimum to a large value
while (there are items to read)
        read next number x from file
        if (x < min)
            min = x
display the minimum
close file
```

File

56
78
93
24
85
63

Solution available on the next page

```
#include <stdio.h>

int main()
{
  FILE *scorefile;
  int score;
  int min;

  scorefile = fopen("scores.txt","r");
  if (scorefile == NULL)
    printf("Error opening input file\n");
  else
    {
      min = 110;
      while (feof(scorefile) == 0) {
        fscanf(scorefile,"%d",&score);
        if (score < min)
          min = score;
      }
    }

  printf("Min = %d\n",min);

  fclose(scorefile);
  system("pause");
  return(0);
}
```

Exercise

- Given a file of integers. Write a program that searches for whether a number appears in the file or not.

- // algorithm to check for y in a file

```
open file
set found to false
while (there are items to read and found is false)
        read next number x from file
        if (x equals y)
            set found to true
            Display found message to user
Display not found message to user
close file
```

File

56
78
93
24
85
63

Solution available on the next page

```
#include <stdio.h>
int main()
{
  FILE *scorefile;
  int score, num, found;

  printf("Please Enter a number\n");
  scanf("%d", &num);

  scorefile = fopen("scores.txt","r");
  if (scorefile == NULL)
    printf("Error opening input file\n");
  else{
       found = 0;
       while ((feof(scorefile) == 0) && (found == 0)) {
         fscanf(scorefile,"%d",&score);
         if (score == num)
            found = 1;
       }
       if (found == 0)
         printf("%d does not appear in the file\n",num);
       else
         printf("%d appears in the file\n",num);
  }
  fclose(scorefile);
  system("pause");
  return(0);
}
```

Home Exercise

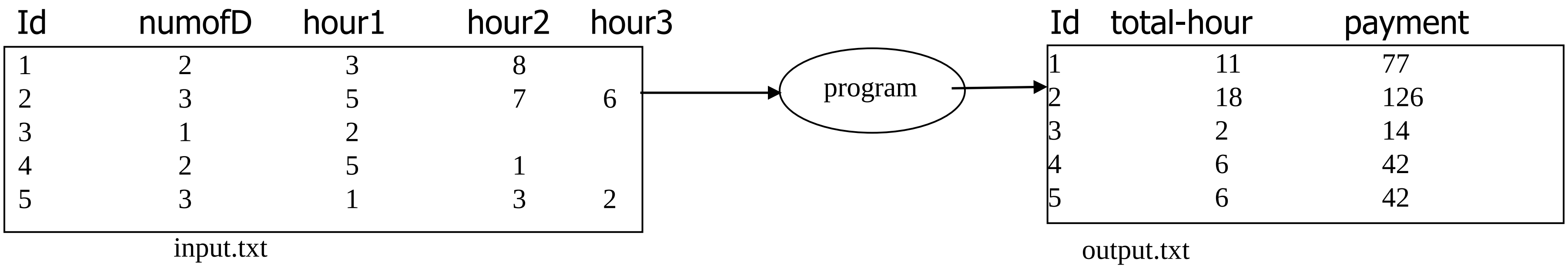
Exercise

- Change the previous program to count how many times the given number appears in the file?

Instead of `fount =1;` put `fount++;`

Read/Write Example

- Suppose we have a data file that contains worker ID, the number of days that a worker worked, and the number of hours the worker worked each day.
- We would like to find out how much to pay for each worker. To compute this, find the total number of hours for each worker and multiply it by 7 dollar/hour.
- For instance, your program should process the following input.txt and generate the corresponding output.txt as follows:



Id	numofD	hour1	hour2	hour3
1	2	3	8	
2	3	5	7	6
3	1	2		
4	2	5	1	
5	3	1	3	2

input.txt

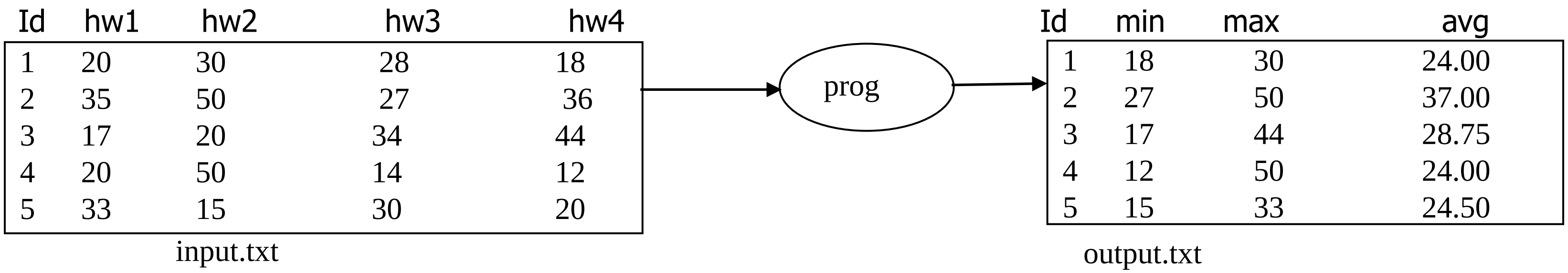
Id	total-hour	payment
1	11	77
2	18	126
3	2	14
4	6	42
5	6	42

output.txt

```
#include <stdio.h>
int main(void)
{
   FILE *infp, *outfp;
   int ID, numofD, hour, i, total_hour;
   if ((infp = fopen("input.txt", "r"))==NULL){
      printf("Input file cannot be opened\n");
      return -1;
   }
   if ((outfp = fopen("output.txt", "w"))==NULL){
      printf("Output file cannot be opened\n");
      return -1;
   }
   while(fscanf(infp, "%d %d",&ID, &numofD)==2) {
       total_hour=0;
       for(i=1; i <= numofD; i++){
         fscanf(infp,”%d”,&hour);
         total_hour +=hour;
      }
      fprintf(outfp, "%3d  %3d   %4d\n",
                      ID, total_hour, total_hour*7);
    }
   fclose(infp);   fclose(outfp);
   return 0;
}
```

Read/write Example

- Suppose we have a data file that contains student ID and his/her homework grades for hw1, hw2, hw3, and hw4.
- We would like to find out min, max and average grade for each student and write this information into another file.
- For instance, your program should process the following input.txt and generate the corresponding output.txt as follows:



Id	hw1	hw2	hw3	hw4
1	20	30	28	18
2	35	50	27	36
3	17	20	34	44
4	20	50	14	12
5	33	15	30	20

input.txt

Id	min	max	avg
1	18	30	24.00
2	27	50	37.00
3	17	44	28.75
4	12	50	24.00
5	15	33	24.50

output.txt

```
#include <stdio.h>
int main(void)
{
   FILE  *infp, *outfp;
   int    ID, hw, max, min;
   double sum;
   if ((infp = fopen("input.txt", "r"))==NULL){
      printf("Input file cannot be opened\n");
      return -1;
   }
   if ((outfp = fopen("output.txt", "w"))==NULL){
      printf("Output file cannot be opened\n");
      return -1;
   }
   while(fscanf(infp, "%d %d",&ID, &hw)==2) {
     sum=max=min=hw;
     for(i=1; i <= 3; i++){
       fscanf(infp,”%d”,&hw);
       sum = sum + hw;
       if (hw > max) max = hw;
       if (hw < min) min = hw;
     }
     fprintf(outfp, "%3d \t %3d \t %4d \t %3.2lf\n",
                      ID, min, max, sum/4);
   }
   fclose(infp);   fclose(outfp);
   return 0;
}
```

Character I/O

- stdio.h has three functions for char I/O

```
int getc(FILE *infp);   /* why return int */
putc(char ch, FILE *outfp);
ungetc(FILE *infp);
```

- File Copy

```
static void CopyFile(FILE *infile, FILE *outfile)
{
    int ch; /* why declare int */
    while ((ch = getc(infile)) != EOF) {
        putc(ch, outfile);
    }
}
```

We could use fscanf(infile, "%c", &ch);

```
static void CopyRemovingComments(FILE *infile, FILE *outfile)
{
    int ch, nch;
    bool commentFlag;
    commentFlag = FALSE;
    while ((ch = getc(infile)) != EOF) {
        if (commentFlag) {
            if (ch == '*') {
                nch = getc(infile);
                if (nch == '/') {
                    commentFlag = FALSE;
                } else {
                    ungetc(nch, infile);
                }
            }
        } else {
            if (ch == '/') {
                nch = getc(infile);
                if (nch == '*') {
                    commentFlag = TRUE;
                } else {
                    ungetc(nch, infile);
                }
            }
            if (!commentFlag) putc(ch, outfile);
        }
    } /* end of while */
}
```

Line-oriented I/O

- stdio.h has two functions for line I/O

```
char *fgets(char buff[], int bsize, FILE *infp);
fputs(char *str, FILE *outfp);
```

- File Copy

```
static void CopyFile(FILE *infile, FILE *outfile)
{
    char buffer[MaxLine+1];

    while (fgets(buffer, MaxLine, infile) != NULL) {
        fputs(buffer, outfile);
    }
}
```

stdin, stdout, stderr

- When a C program starts, it opens three files with the following file pointers:

`stdin` →keyboard,

`stdout` →screen,

`stderr` →screen

- `stdin` and `stdout` might be redirected

```
main212> myprog < infile  > outfile
```

File copy using indirections

- #define getchar()   getc(stdin)
- #define putchar(c)   putc((c), stdout)

```
/* version 1 */
#include <stdio.h>
main()
{
  int c;

  c = getchar();
  while (c != EOF){
    putchar(c);
    c = getchar();
  }
}
```

```
/* version 2 */
#include <stdio.h>
main()
{
  int c;

  while ((c = getchar()) != EOF){
    putchar(c);
  }
}
```

```
…> myprog < infile.txt > outfile.txt
```

Textbook’s simpio.h

- `int GetInteger(void);`
- `long GetLong(void);`
- `double GetReal(void);`
- `string GetLine(void);`
- **`string ReadLine(FILE *infile);`**

  `/* dynamically allocates memory */`
- `#define GetLine(void) Readline(stdin)`

Standard C I/O (stdio)

clearerr()
fclose()
feof()
ferror()
fflush()
fgetc()
fgetpos()
fgets()
fopen()
fprintf()
fputc()
fputs()
fread()
freopen()
fscanf()
fseek()
fsetpos()
ftell()
fwrite()
getc()
getchar()
gets()
perror()
printf()
putc()
putchar()
puts()
remove()
rename()
rewind()
scanf()
setbuf()
setvbuf()
sprintf()
sscanf()
tmpfile()
tmpnam()
ugetc()
vfprintf()
vprintf()
vsprintf()

fflush(): If the given file stream is an output stream, then fflush() causes the output buffer to be written to the file. If the given stream is of the input type, then fflush() causes the input buffer to be cleared.

Text Files vs. Binary Files

http://www.fileinfo.com/help/binary_vs_text_files

- The two file types may look the same on the surface, but they encode data differently. While both binary and text files contain data stored as a series of bits (binary values of 1s and 0s), the bits in text files represent characters, while the bits in binary files represent custom data.
- Binary files typically contain a sequence of bytes, or ordered groupings of eight bits. When creating a custom file format for a program, a developer arranges these bytes into a format that stores the necessary information for the application. Binary file formats may include multiple types of data in the same file, such as image, video, and audio data. This data can be interpreted by supporting programs, but will show up as garbled text in a text editor.
- Text files are more restrictive than binary files since they can only contain textual data. However, unlike binary files, they are less likely to become corrupted. While a small error in a binary file may make it unreadable, a small error in a text file may simply show up once the file has been opened.
- We just discussed text files….

Other Library Functions

Math Functions

```
#include <math.h>
```

`fabs(x)`	Absolute value of `x`.
`sqrt(x)`	Square root of `x`, where `x`>=0.
`pow(x,y)`	Exponentiation, x^y. Errors occur if `x`=0 and `y`<=0, or if `x`<0 and `y` is not an integer.
`ceil(x)`	Rounds `x` to the nearest integer toward ∞ (infinity). Example, `ceil(2.01)` is equal to 3.
`floor(x)`	Rounds `x` to the nearest integer toward -∞ (negative infinity). Example, `floor(2.01)` is equal to 2.
`exp(x)`	Computes the value of e^x.
`log(x)`	Returns ln x, the natural logarithm of `x` to the base *e*. Errors occur if `x`<=0.
`log10(x)`	Returns log10x, logarithm of x to the base 10. Errors occur if `x`<=0.

Trigonometric Functions

sin(x) Computes the sine of x, where x is in radians.
cos(x) Computes the cosine of x, where x is in radians

tan(x) Computes the tangent of x, where x is in radians.
asin(x) Computes the arcsine or inverse sine of x,
where x must be in the range [-1, 1].
Returns an angle in radians in the range [-π/2,π/2].
acos(x) Computes the arccosine or inverse cosine of x,
where x must be in the range [-1, 1].
Returns an angle in radians in the range [0, π].
atan(x) Computes the arctangent or inverse tangent of x. The
Returns an angle in radians in the range [-π/2,π/2].
atan2(y,x) Computes the arctangent or inverse tangent of the value
y/x. Returns an angle in radians in the range [-π, π].

Meaning of Parameters of a function

- A function may contain no argument or contain one or more arguments
- If more than one argument, list the arguments in the correct order
- Be careful about the meaning of an argument. For example, sin(x) assumes that x is given in radians, so to compute the sin of 60 degree, you need to first conver 60 degree into radian then call sin function:

```
#define PI   3.141593
theta = 60;
theta_rad = theata * PI / 180;
b = sin(theta_rad);        /* is not the same as  sin(theta); */
```

Exercise

- Write an expression to compute velocity using the following equation
- Assume that the variables are declared

$$velocity = \sqrt{vo^2 + 2a(x - xo)}$$

```
velocity = sqrt(vo*vo+2*a*(x-xo));
```

```
velocity = sqrt(pow(vo,2)+2*a*(x-xo));
```

Exercise

- Write an expression to compute velocity using the following equation
- Assume that the variables are declared

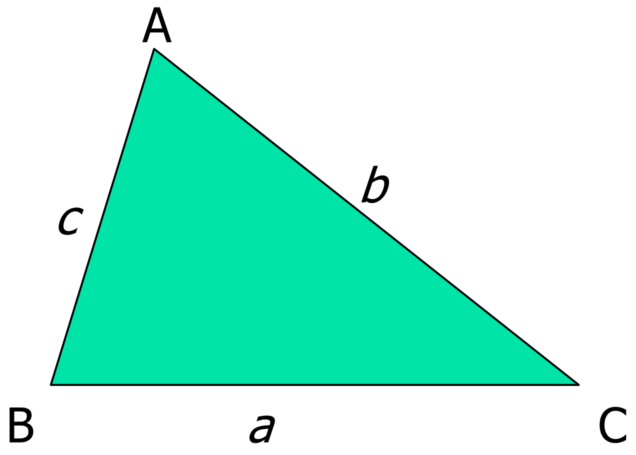
$$center = \frac{38.19(r^3 - s^3)\sin a}{(r^2 - s^2)a}$$

Make sure that *a* is given in radian; otherwise, first convert it to radian

```
center = (38.19*(pow(r,3)-pow(s,3))*sin(a))/
                         ((pow(r,2)-pow(s,2))*a);
```

```
center = (38.19*(r*r*r - s*s*s)*sin(a))/((r*r –s*s)*a);
```

Exercise



Write a program that asks user to enter A in degrees, *a* and *b* in cm, then computes

B=?	in degrees
C=?	in degrees
c=?	in cm
area=?	in cm^2

For example, given A=36^o, *a*=8 cm, *b*=5 cm: B=21.55^o, C=122.45^o, *c*=11.49 cm

$$\frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C}$$

$$area = \frac{1}{2} ab \sin C = \frac{1}{2} ac \sin B = \frac{1}{2} bc \sin A$$

Character Functions

```
#include <ctype.h>
int ch;                /* why int */
putchar(‘a’);
ch = getchar();
```

Alphanumeric	isalnum(ch)
alphabetic	isalpha(ch)
control character	iscntrl(ch)
decimal digit	isdigit(ch)
printing character (not incl space)	isgraph(ch)
lower case letter	islower(ch)
printing character (incl space)	isprint(ch)
printing char except space, letter, digit?	ispunct(ch)
space, formfeed, newline, cr, tab, vtab?	isspace(ch)
upper case letter	isupper(ch)
hexadecimal digit	isxdigit(ch)
convert to lower case	tolower(ch)
convert to upper case	toupper(ch)

Exercise

What is the output of the following program

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

int main(void)
{
  char ch1='a', ch2;
  char ch3='X', ch4;
  char ch5='8';

  ch2 = toupper(ch1);
  printf("%c %c \n",ch1,ch2);
  ch4 = tolower(ch3);
  printf("%c %c \n",ch3,ch4);

  printf("%d\n",isdigit(ch5));
  printf("%d\n",islower(ch1));
  printf("%d\n",isalpha(ch5));

  return(0);
}
```