

DASF004

Basic and Practice in Programming

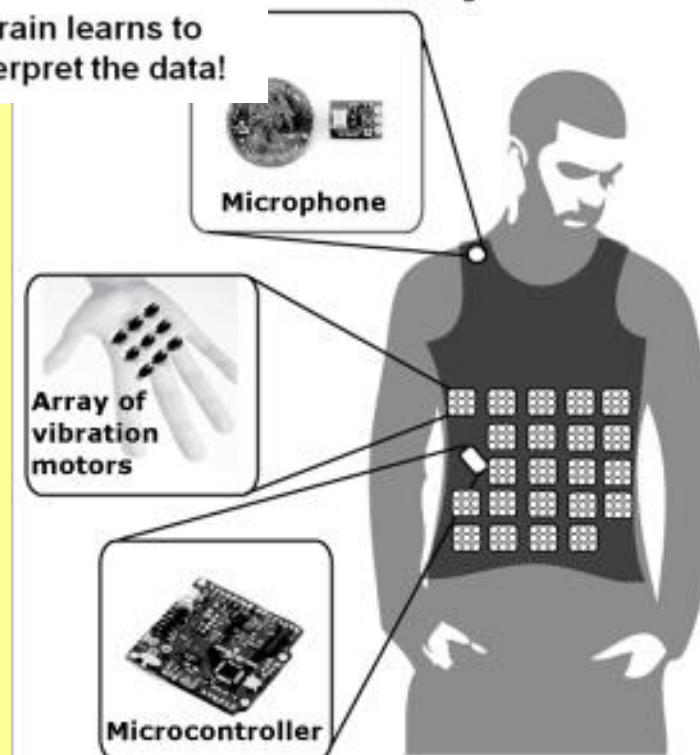
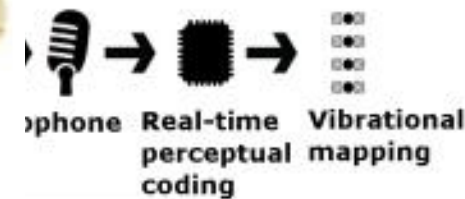
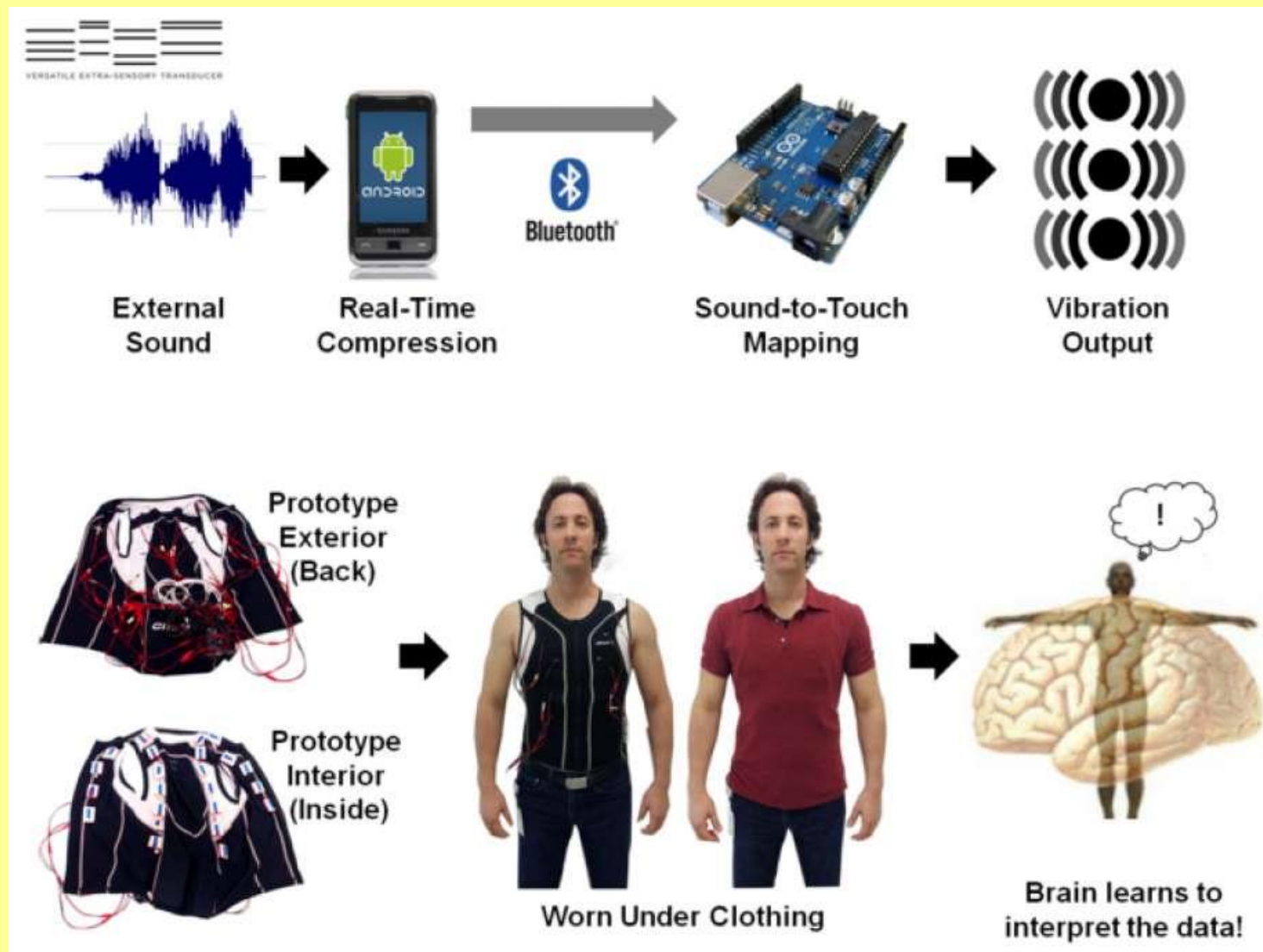
Lecture 11

File Processing

Administration

- Final exam
 - The Final Assignment will be similar to the Midterm Assignment
 - It will be available on icampus
 - Tuesday 1 June 2020, 12:00 pm (noon)
 - Due date: Wednesday 2 June 2020 18:00 pm

Sensory Substitution – Extrasensation ...



Agenda

File Processing

- Input: How do you take input from a file
- Output: How do you generate output to a file

Two kinds of file access

Sequential File Access

Random File Access

File Access

There are two kinds of file access

1. Sequential Access

- You have to read and write the file starting from the beginning, reading and writing byte by byte from the start of the file

2. Random Access

- You can read and write any byte in the file

Files

- C views each file simply as a sequential stream of bytes (Fig. 11.1).
- Each file ends either with an **end-of-file marker** (Ctrl-Z, denoted by ^Z) or at a specific byte number recorded in a system-maintained, administrative data structure.



Fig. 11.1 | C's view of a file of n bytes.

```

1 // Fig. 11.2: fig11_02.c
2 // Creating a sequential file
3 #include <stdio.h>
4
5 int main( void )
6 {
7     unsigned int account; // account number
8     char name[ 30 ]; // account name
9     double balance; // account balance
10
11     FILE *cfPtr; // cfPtr = clients.dat file pointer
12
13     // fopen opens file. Exit program if unable to create file
14     if ( ( cfPtr = fopen( "clients.dat", "w" ) ) == NULL ) {
15         puts( "File could not be opened" );
16     } // end if
17     else {
18         puts( "Enter the account, name, and balance." );
19         puts( "Enter EOF to end input." );
20         printf( "%s", "? " );
21         scanf( "%d%29s%1f", &account, name, &balance );
22
23         // write account, name and balance into file with fprintf
24         while ( !feof( stdin ) ) {
25             fprintf( cfPtr, "%d %s %.2f\n", account, name, balance );
26             printf( "%s", "? " );
27             scanf( "%d%29s%1f", &account, name, &balance );
28         } // end while
29
30         fclose( cfPtr ); // fclose closes file
31     } // end else
32 } // end main

```

Enter the account, name, and balance.

Enter EOF to end input.

? **100 Jones 24.98**

? **200 Doe 345.67**

? **300 White 0.00**

? **400 Stone -42.16**

? **500 Rich 224.62**

? **^Z**

File Access – 4 Steps

1. Declare a file pointer

```
FILE *fPtr;
```

2. Open file stream (write only)

```
fPtr = fopen("filename", "w");
```

3. Access the file stream (writing a file)

```
fprintf(fPtr, "Hello, World");
```

4. Close the file stream

```
fclose(fPtr);
```

Example: Writing file

```
#include <stdio.h>

int main(void)
{
    FILE *fPtr;
    fPtr = fopen("testfile.txt", "w");
    fprintf(fPtr, "Hello, World!");
    fclose(fPtr);

    return 0;
}
```

File Access – 4 Steps

1. Declare a file pointer

```
FILE *fPtr;
```

2. Open file stream (read only)

```
fPtr = fopen("filename", "r");
```

3. Access the file stream (reading a file)

```
fscanf(fPtr, "%s", &input);
```

4. Close the file stream

```
fclose(fPtr);
```

Example: Reading file

```
#include <stdio.h>

int main(void)
{ FILE *fPtr;
  char input[20];

  fPtr = fopen("testfile.txt", "r");
  fscanf(fPtr, "%s", input);
  printf("%s", input);
  fclose(fPtr);

  return 0;
}
```

Note on `fscanf()` function

- `fscanf()` is similar to `scanf()`.
- It will read characters until a space, tab, newline or end-of-file indicator is encountered. `#include <stdio.h>`
- In last example:
 - The file contains the string “Hello, World!”
 - `fscanf()` only read in “Hello,”

Example: Reading file 2

```
#include <stdio.h>

int main(void)
{
    FILE *fPtr;
    char input[20];

    fPtr = fopen("testfile.txt", "r");
    fscanf(fPtr, "%s", &input);
    printf("%s", input);
    fscanf(fPtr, "%s", &input);
    printf("%s", input);
    fclose(fPtr);

    return 0;
}
```

Asking the user to enter and then write to file continuously (sequential access)?

- Using a loop (e.g. while loop)

```
Enter a value (or -1 to end the input): 69
Enter a value (or -1 to end the input): 72
Enter a value (or -1 to end the input): 84
Enter a value (or -1 to end the input): 89
Enter a value (or -1 to end the input): 68
Enter a value (or -1 to end the input): 95
Enter a value (or -1 to end the input): 99
Enter a value (or -1 to end the input): 98
Enter a value (or -1 to end the input): 76
Enter a value (or -1 to end the input): -1
```



numbers.txt - Notepad

File Edit Format View Help

```
69 72 84 89 68 95 99 98 76
```

Example: Writing to file with a loop

```
#include <stdio.h>

int main(void)
{ FILE *fPtr;
  int input;

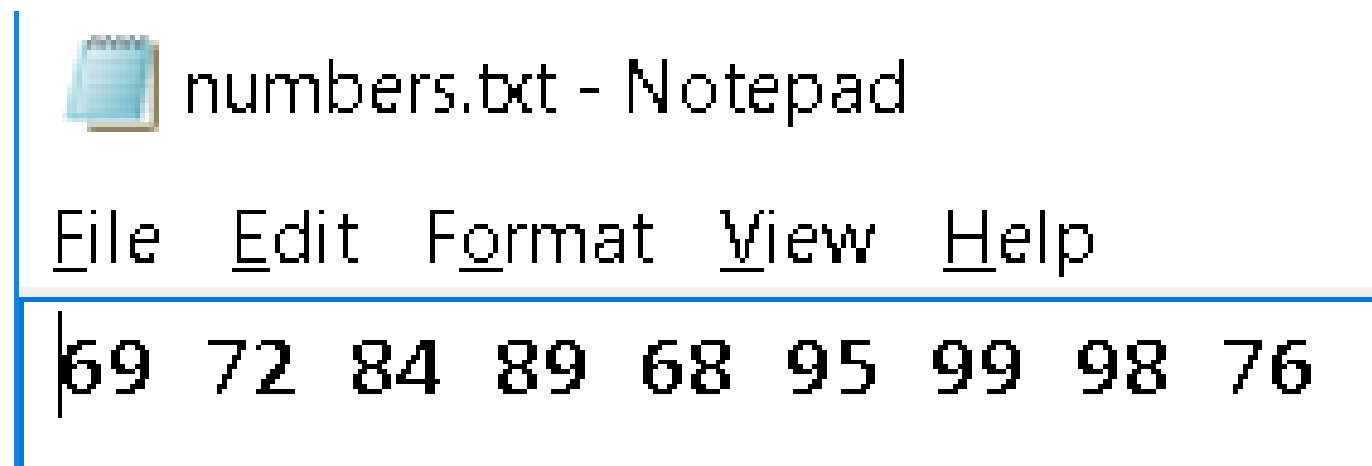
  fPtr = fopen("numbers.txt", "w");

  printf("Enter a value (or -1 to end the input): ");
  scanf("%d", &input);
  while(input != -1)
  { fprintf(fPtr, "%d ", input);
    printf("Enter a value (or -1 to end the input): ");
    scanf("%d", &input);
  }

  fclose(fPtr);
  return 0;
}
```


How do you read a file continuously?

- Using a loop (e.g. do ... while loop)
 - e.g. if you have the following file:



```
numbers.txt - Notepad
File Edit Format View Help
69 72 84 89 68 95 99 98 76
```

- You are to read in a sequence of integers
- You don't know how many integers are there in the file
- And you want to read this file as input
- The loop breaks when it reach the [end-of-file marker](#)

Example: Reading file continuously with a loop

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

```
int main(void)
```

```
{ FILE *fPtr;
```

```
  int input;
```

```
  fPtr = fopen("numbers.txt", "r");
```

```
  do
```

```
  { fscanf(fPtr, "%d", &input);
```

```
    printf("%d ", input);
```

```
  } while(!feof(fPtr));
```

```
  fclose(fPtr);
```

```
  return 0;
```

```
}
```



C:\Users\Arthur Tang\Documents\Untitled1.exe

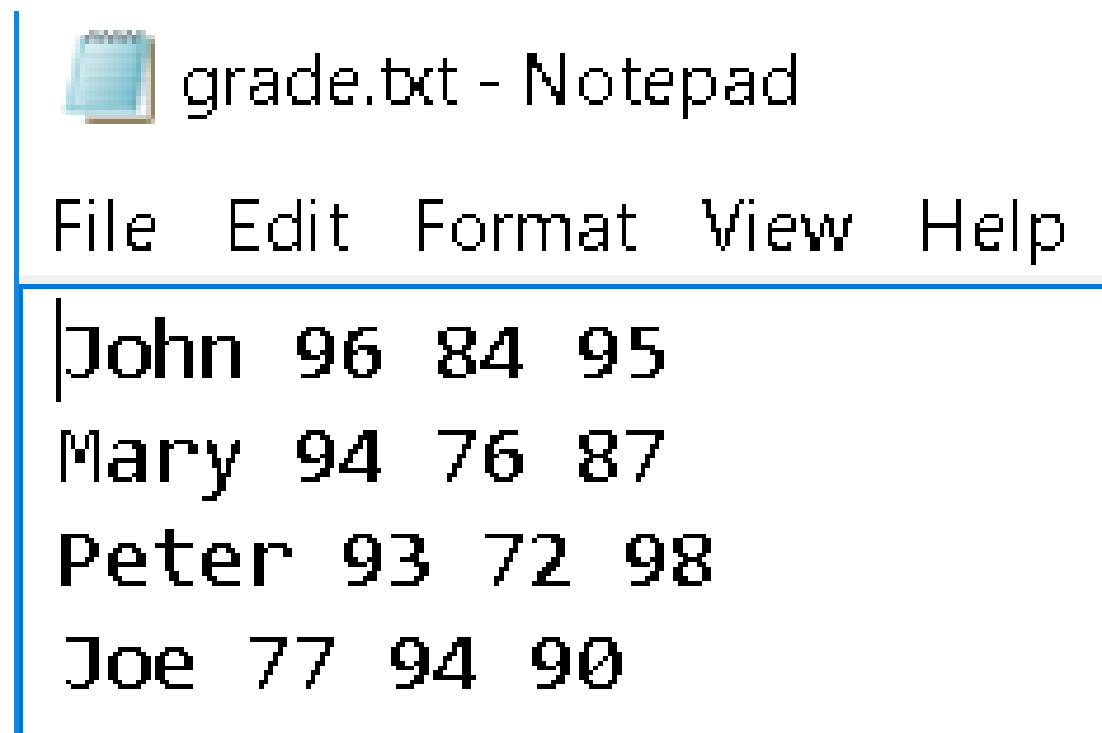
69 72 84 89 68 95 99 98 76 76

Process exited after 0.01393 seconds with return value 0

Press any key to continue . . .

A slightly more complex example ...

- You are reading from a file sequentially (sequential access)
- You know ahead of time the data format
 - One string, followed by three integers in each line



```
grade.txt - Notepad
File Edit Format View Help
John 96 84 95
Mary 94 76 87
Peter 93 72 98
Joe 77 94 90
```

- You don't know how many lines are there in the file

Example: Reading file continuously with a loop

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

```
int main(void)
```

```
{ FILE *fPtr;
```

```
  int number[3];
```

```
  char name[20];
```

```
  fPtr = fopen("grade.txt", "r");
```

```
  fscanf(fPtr, "%s", &name);
```

```
  while(!feof(fPtr))
```

```
  { fscanf(fPtr, "%d %d %d", &number[0], &number[1], &number[2]);
```

```
    printf("%s %d %d %d\n", name, number[0], number[1], number[2]);
```

```
    fscanf(fPtr, "%s", &name);
```

```
  }
```

```
  fclose(fPtr);
```

```
  return 0;
```

```
}
```



C:\Users\Arthur Tang\Documents\Untitled1.exe

John 96 84 95

Mary 94 76 87

Peter 93 72 98

Joe 77 94 90

Process exited after 0.01092 seconds with return value 0

Press any key to continue . . .

Problems ...

- What if the file you are going to write on already exists???
 - You will overwrite the file that exists
 - If an existing file is opened for writing, the contents of the file are *discarded without warning*.
- You need to be very careful about this ...
- What if the file you are going to read from does not exist???
 - Logical error: Your program will behave unpredictably...
- So how do you fix this problem???

Return value of `fopen()`

- Opening a file for read:

```
fPrt = fopen("grade.txt", "r");
```

- If the file does not exist, `fopen()` returns a NULL pointer

Example: Reading file continuously with a loop

```
#include <stdio.h>

int main(void)
{ FILE *fPtr;
  int number[3];
  char name[20];

  fPtr = fopen("grade.txt", "r");

  if(fPtr != NULL) // Check if the open is successful
  { fscanf(fPtr, "%s", &name);
    while(!feof(fPtr))
    { fscanf(fPtr, "%d %d %d", &number[0], &number[1], &number[2]);
      printf("%s %d %d %d\n", name, number[0], number[1], number[2]);
      fscanf(fPtr, "%s", &name);
    }
  }
  fclose(fPtr);

  return 0;
}
```

 C:\Users\Arthur Tang\Documents\Untitled1.exe

John 96 84 95

Mary 94 76 87

Peter 93 72 98

Joe 77 94 90

Process exited after 0.01092 seconds with return value 0

Press any key to continue . . .

Another way ...

```
#include <stdio.h>

int main(void)
{ FILE *fPtr;
  int number[3];
  char name[20];

  if ((fPtr=fopen("grade.txt","r")) != NULL) // Check if the open is good
  { fscanf(fPtr,"%s",&name);
    while(!feof(fPtr))
    { fscanf(fPtr,"%d %d %d",&number[0],&number[1],&number[2]);
      printf("%s %d %d %d\n",name,number[0],number[1],number[2]);
      fscanf(fPtr,"%s",&name);
    }
  }
  fclose(fPtr);

  return 0;
}
```

 C:\Users\Arthur Tang\Documents\Untitled1.exe

John 96 84 95

Mary 94 76 87

Peter 93 72 98

Joe 77 94 90

Process exited after 0.01092 seconds with return value 0

Press any key to continue . . .

Mode	Description
r	Open an existing file for reading.
w	Create a file for writing. If the file already exists, discard the current contents.
a	Append: open or create a file for writing at the end of the file.
r+	Open an existing file for update (reading and writing).
w+	Create a file for update. If the file already exists, discard the current contents.
a+	Append: open or create a file for update; writing is done at the end of the file.
rb	Open an existing file for reading in binary mode.
wb	Create a file for writing in binary mode. If the file already exists, discard the current contents.
ab	Append: open or create a file for writing at the end of the file in binary mode.
rb+	Open an existing file for update (reading and writing) in binary mode.
wb+	Create a file for update in binary mode. If the file already exists, discard the current contents.
ab+	Append: open or create a file for update in binary mode; writing is done at the end of the file.

11.3 Creating a Sequential-Access File (Cont.)

- `r` – open file for read only
- `w` – open file for write only; if the file exists, then overwrite the existing file
- `r+` – open file for read and write; if the file exists, then don't write anything and discard all information
- `w+` – open file for read and write; if the file exists, then overwrite the existing file

11.3 Creating a Sequential-Access File (Cont.)

- a – Open file and append information at the end of the file; if the file does not exist, create a new file
- a+ – Open file for reading and append; if the file does not exist, create a new file

Access Mode

	r	w	a	r+	w+	a+
file must exist before open	y			y		
old file deleted		y			y	
file can be read	y			y	y	y
file can be written		y	y	y	y	y
file written only at end			y			y

r: read w: write a: append

You can open multiple files access...

```
#include <stdio.h>

int main(void)
{ FILE *f1Ptr, *f2Ptr, *f3Ptr;
  char name[30];
  fPtr1 = fopen("StudentsInfo.txt", "r");
  fPtr2 = fopen("grade.txt", "r+");
  fPtr3 = fopen("report.txt", "w+");
  fscanf(f1Ptr, "%s", &name);

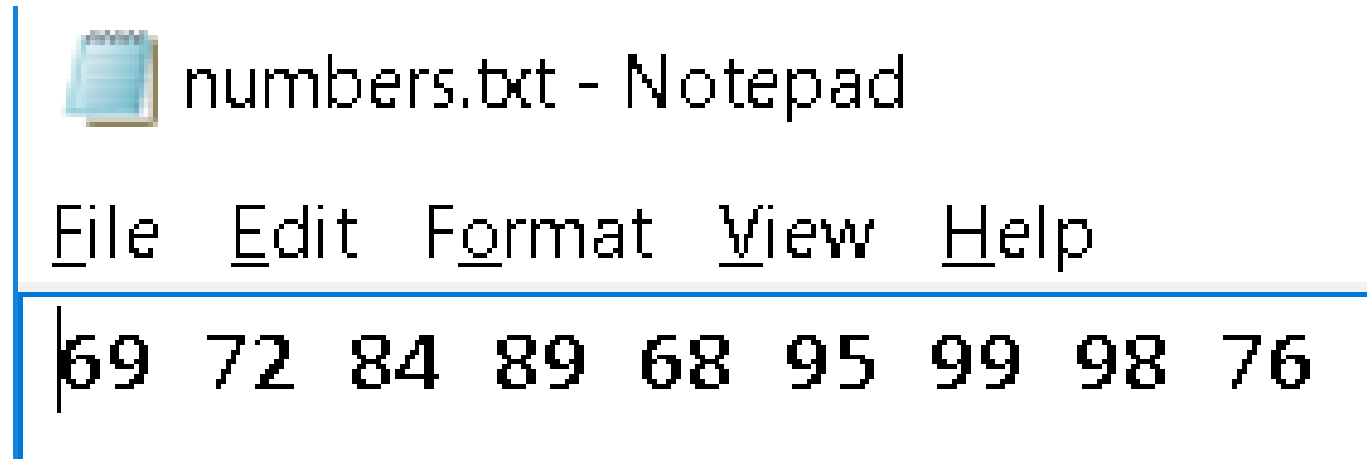
  ... ..

  fclose(f1Ptr);
  fclose(f2Ptr);
  fclose(f3Ptr);
  return 0;
}
```

Reading Data from Sequential Access File

- So far you are reading the file input byte by byte

- e.g.



```
numbers.txt - Notepad
File Edit Format View Help
69 72 84 89 68 95 99 98 76
```

- In previous example, 69, 72, 84, 89, 68, 95, 99, 98, 76 are read in sequence.
- Issue: What if you need to go back?
- In some occasion, you may need to read the file (starting from the beginning) several time.

`rewind()` function

- The statement

- `rewind(fPtr);`

causes a program's **file position pointer**—which indicates the number of the next byte in the file to be read or written—to be repositioned to the *beginning* of the file (i.e., byte 0) pointed to by `fPtr`.

rewind()

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

```
int main(void)
```

```
{ FILE *fPtr;
```

```
  int x[9];
```

```
  fPtr = fopen("numbers.txt", "r");
```

```
  fscanf(fPtr, "%d %d %d %d", &x[0], &x[1], &x[2], &x[3]);
```

```
  printf("1: %d %d %d %d\n", x[0], x[1], x[2], x[3]);
```

```
  rewind(fPtr);
```

```
  fscanf(fPtr, "%d %d %d %d %d %d", &x[0], &x[1], &x[2], &x[3], &x[4], &x[5]);
```

```
  printf("2: %d %d %d %d %d %d\n", x[0], x[1], x[2], x[3], x[4], x[5]);
```

```
  rewind(fPtr);
```

```
  fscanf(fPtr, "%d %d", &x[0], &x[1]);
```

```
  printf("3: %d %d\n", x[0], x[1]);
```

```
  fclose(fPtr);
```

```
  return 0;
```

```
}
```



numbers.txt - Notepad

File Edit Format View Help

69 72 84 89 68 95 99 98 76



C:\Users\Arthur Tang\Documents\Untitled3.exe

1: 69 72 84 89

2: 69 72 84 89 68 95

3: 69 72

Process exited after 0.01863 seconds with return value 0
Press any key to continue . . .

11.4 Reading Data from a Sequential-Access File (Cont.)

Credit Inquiry Program

- ▶ The program of Fig. 11.7 allows a credit manager to obtain lists of customers with zero balances (i.e., customers who do not owe any money), customers with credit balances (i.e., customers to whom the company owes money) and customers with debit balances (i.e., customers who owe the company money for goods and services received).
- ▶ A credit balance is a *negative* amount; a debit balance is a *positive* amount.

11.4 Reading Data from a Sequential-Access File (Cont.)

The program displays a menu and allows the credit manager to enter one of three options to obtain credit information.

Option 1 produces a list of accounts with zero balances.

Option 2 produces a list of accounts with *credit balances*.

Option 3 produces a list of accounts with *debit balances*.

Option 4 terminates program execution.

A sample output is shown in Fig. 11.8.

Enter request

- 1 - List accounts with zero balances
- 2 - List accounts with credit balances
- 3 - List accounts with debit balances
- 4 - End of run

? 1

Accounts with zero balances:

300	White	0.00
-----	-------	------

? 2

Accounts with credit balances:

400	Stone	-42.16
-----	-------	--------

? 3

Accounts with debit balances:

100	Jones	24.98
-----	-------	-------

200	Doe	345.67
-----	-----	--------

500	Rich	224.62
-----	------	--------

? 4

End of run.



Untitled - Notepad

File	Edit	Format	View	Help
------	------	--------	------	------

100	Jones	24.98
-----	-------	-------

200	Doe	345.67
-----	-----	--------

300	White	0.00
-----	-------	------

400	Stone	-42.16
-----	-------	--------

500	Rich	244.62
-----	------	--------

Fig. 11.8 | Sample output of the credit inquiry program of Fig. 11.7.

[illegible]

```

1 // Fig. 11.7: fig11_07.c
2 // Credit inquiry program
3 #include <stdio.h>
4
5 // function main begins program execution
6 int main( void )
7 {
8     unsigned int request; // request number
9     unsigned int account; // account number
10    double balance; // account balance
11    char name[ 30 ]; // account name
12    FILE *cfPtr; // clients.dat file pointer
13
14    // fopen opens the file; exits program if file cannot be opened
15    if ( ( cfPtr = fopen( "clients.dat", "r" ) ) == NULL ) {
16        puts( "File could not be opened" );
17    } // end if

```

```

        if(file open failed) // --- Part 1
        { error message
        }

        else // Main Body of the Program!!! [Part 2 - Part 6]
        { Display Menu // --- Part 2

            while(request not 4)
            { Read 3 variables from file

                switch(request)
                { case 1: // --- Part 3
                    Read all record from file with loop
                    Display record if balance == 0
                    break;
                    case 2: // --- Part 4
                    Read all record from file with loop
                    Display record if balance < 0
                    break;
                    case 3: // --- Part 5
                    Read all record from file with loop
                    Display record if balance > 0
                    break;
                }
                rewind file pointer
                Ask for request
            }
            Close file // --- Part 6
            End of Program
        }

```

Fig. 11.7 | Credit inquiry program. (Part 1 of 6.)

```

18  else {
19
20      // display request options
21      printf( "%s", "Enter request\n"
22              " 1 - List accounts with zero balances\n"
23              " 2 - List accounts with credit balances\n"
24              " 3 - List accounts with debit balances\n"
25              " 4 - End of run\n? " );
26      scanf( "%u", &request );
27
28      // process user's request
29      while ( request != 4 ) {
30
31          // read account, name and balance from file
32          fscanf( cfPtr, "%d%29s%lf", &account, name, &balance );
33
                                     if(file open failed)                // --- Part 1
                                     { error message
                                     }
                                     else // Main Body of the Program!!!    [Part 2 - Part 6]
                                     { Display Menu                        // --- Part 2

                                     while(request not 4)
                                     { Read 3 variables from file

                                     switch(request)
                                     { case 1:                                // --- Part 3
                                         Read all record from file with loop
                                         Display record if balance == 0
                                         break;
                                     case 2:                                // --- Part 4
                                         Read all record from file with loop
                                         Display record if balance < 0
                                         break;
                                     case 3:                                // --- Part 5
                                         Read all record from file with loop
                                         Display record if balance > 0
                                         break;
                                     }
                                     rewind file pointer
                                     Ask for request
                                     }
                                     Close file                                // --- Part 6
                                     End of Program
                                     }

```

Fig. 11.7 | Credit inquiry program. (Part 2 of 6.)

```

34 switch ( request ) {
35     case 1:
36         puts( "\nAccounts with zero balances:" );
37
38         // read file contents (until eof)
39         while ( !feof( cfPtr ) ) {
40
41             if ( balance == 0 ) {
42                 printf( "%-10d%-13s%7.2f\n",
43                     account, name, balance );
44             } // end if
45
46             // read account, name and balance from file
47             fscanf( cfPtr, "%d%29s%lf",
48                 &account, name, &balance );
49         } // end while
50
51         break;

```

```

if(file open failed) // --- Part 1
{ error message
}
else // Main Body of the Program!!! [Part 2 - Part 6]
{ Display Menu // --- Part 2

while(request not 4)
{ Read 3 variables from file

```

Fig. 11.7 | Credit inquiry program. (Part 3 of 6.)

```

switch(request)
{ case 1: // --- Part 3
    Read all record from file with loop
    Display record if balance == 0
    break;

    case 2: // --- Part 4
        Read all record from file with loop
        Display record if balance < 0
        break;

    case 3: // --- Part 5
        Read all record from file with loop
        Display record if balance > 0
        break;

    }
    rewind file pointer
    Ask for request
}
Close file // --- Part 6
End of Program
}

```

```

52 case 2:
53     puts( "\nAccounts with credit balances:\n" );
54
55     // read file contents (until eof)
56     while ( !feof( cfPtr ) ) {
57
58         if ( balance < 0 ) {
59             printf( "%-10d%-13s%7.2f\n",
60                 account, name, balance );
61         } // end if
62
63         // read account, name and balance from file
64         fscanf( cfPtr, "%d%29s%1f",
65             &account, name, &balance );
66     } // end while
67
68     break;

```

Fig. 11.7 | Credit inquiry program. (Part 4 of 6.)

```

if(file open failed) // --- Part 1
{ error message
}
else // Main Body of the Program!!! [Part 2 - Part 6]
{ Display Menu // --- Part 2

while(request not 4)
{ Read 3 variables from file

switch(request)
{ case 1: // --- Part 3
    Read all record from file with loop
    Display record if balance == 0
    break;

    case 2: // --- Part 4
    Read all record from file with loop
    Display record if balance < 0
    break;

    case 3: // --- Part 5
    Read all record from file with loop
    Display record if balance > 0
    break;

    }
    rewind file pointer
    Ask for request
}
Close file // --- Part 6
End of Program
}

```



```

69         case 3:
70             puts( "\nAccounts with debit balances:\n" );
71
72             // read file contents (until eof)
73             while ( !feof( cfPtr ) ) {
74
75                 if ( balance > 0 ) {
76                     printf( "%-10d%-13s%7.2f\n",
77                         account, name, balance );
78                 } // end if
79
80                 // read account, name and balance from file
81                 fscanf( cfPtr, "%d%29s%1f",
82                     &account, name, &balance );
83             } // end while
84
85             break;
86         } // end switch
87
88         rewind( cfPtr ); // return cfPtr to b
89
90         printf( "%s", "\n? " );
91         scanf( "%d", &request );
92     } // end while
93
94     if(file open failed) // --- Part 1
95     { error message
96     }
97     else // Main Body of the Program!!! [Part 2 - Part 6]
98     { Display Menu // --- Part 2
99
100         while(request not 4)
101         { Read 3 variables from file
102
103             switch(request)
104             { case 1: // --- Part 3
105                 Read all record from file with loop
106                 Display record if balance == 0
107                 break;
108             case 2: // --- Part 4
109                 Read all record from file with loop
110                 Display record if balance < 0
111                 break;
112             case 3: // --- Part 5
113                 Read all record from file with loop
114                 Display record if balance > 0
115                 break;
116             }
117             rewind file pointer
118             Ask for request
119         }
120         Close file // --- Part 6
121         End of Program
122     }

```

Fig. 11.7 | Credit inquiry program. (Part 5 of 6.)

```

94         puts( "End of run." );
95         fclose( cfPtr ); // fclose closes the file
96     } // end else
97 } // end main

```

Fig. 11.7 | Credit inquiry program. (Part 6 of 6.)

```

if(file open failed) // --- Part 1
{ error message
}
else // Main Body of the Program!!! [Part 2 - Part 6]
{ Display Menu // --- Part 2

    while(request not 4)
    { Read 3 variables from file

        switch(request)
        { case 1: // --- Part 3
            Read all record from file with loop
            Display record if balance == 0
            break;
          case 2: // --- Part 4
            Read all record from file with loop
            Display record if balance < 0
            break;
          case 3: // --- Part 5
            Read all record from file with loop
            Display record if balance > 0
            break;
        }
        rewind file pointer
        Ask for request
    }
    Close file // --- Part 6
    End of Program
}

```

11.4 Reading Data from a Sequential-Access File (Cont.)

What if you want to add entry in the file???

What if you want to modify the entry in the file???

Data in this type of sequential file cannot be modified without the risk of destroying other data.

For example, if the name “white” needs to be changed to “worthington,” the old name cannot simply be overwritten.

The record for white was written to the file as

11.4 Reading Data from a Sequential-Access File (Cont.)

If the record is rewritten beginning at the same location in the file using the new name, the record will be

- 300 Worthington 0.00

The new record is larger (has more characters) than the original record.

The characters beyond the second “o” in “Worthington” will *overwrite* the beginning of the next sequential record in the file.

The problem here is that in the [formatted input/output model](#) using `fprintf` and `fscanf`, fields—and hence records—can vary in size.



Untitled - Notepad

File Edit Format View Help

100	Jones	24.98
200	Doe	345.67
300	White	0.00
400	Stone	-42.16
500	Rich	244.62

11.4 Reading Data from a Sequential-Access File (Cont.)

For example, the values 7, 14, -117, 2074 and 27383 are all `ints` stored in the same number of bytes internally, but they're different-sized fields when displayed on the screen or written to a file as text. Therefore, sequential access with `fprintf` and `fscanf` is *not* usually used to *update records in place*.

Instead, the entire file is usually *rewritten*.

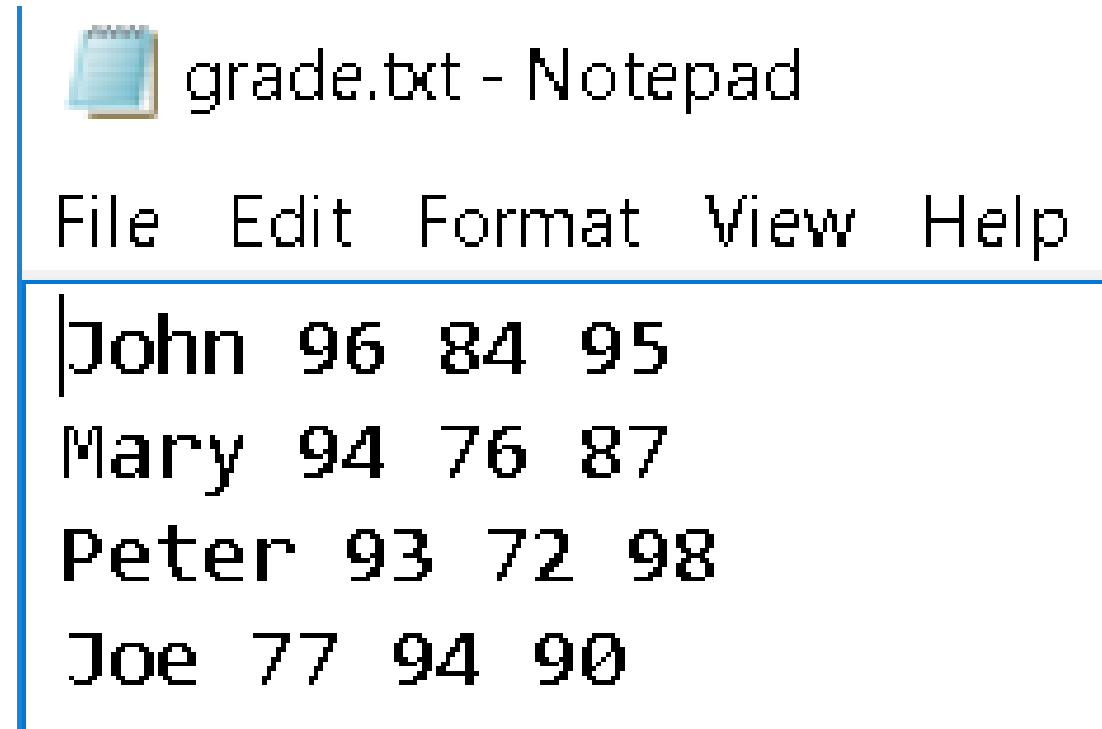
11.4 Reading Data from a Sequential-Access File (Cont.)

To make the preceding name change, the records before 300 white 0.00 in such a sequential-access file would be copied to a new file, the new record would be written and the records after 300 white 0.00 would be copied to the new file.

This requires processing every record in the file to update one record.

Data Structure and Representation in File Input/Output

- When you read data from a file, you need to know the input file data format
- e.g.



```
grade.txt - Notepad
File Edit Format View Help
John 96 84 95
Mary 94 76 87
Peter 93 72 98
Joe 77 94 90
```

- Each line is representing a student
- The first item in each line is the student's name
- The second item in each line is the first score for the student
- The third item in each line is the second score for the student
- The fourth item in each line is the third score for the student
- Then you can design your logic to read in the data, and then perform the processing

Binary Records

- So far we've been dealing with records in ascii form (i.e. text form)
 - You can view the records in any text editor (such as notepad)
- Binary Records
 - Records are in 1 and 0 only (i.e. not in text form)
 - They cannot be viewed in notepad
- You can use the function `fwrite()` to write to a file in binary form

ASCII vs. Binary data

Data: 10 20

ASCII:

- Characters “1” “0” “ ” “2” “0”

- “0”: 00110000

- “1”: 00110001

- “2”: 00110010

- “ ”: 00100000

- “10 20”

- 00110001 00110000 00100000 00110010 00110000

Binary:

- 10: 00000000 00000000 00000000 00001010

- 20: 00000000 00000000 00000000 00010100

- 10 20

- 00000000 00000000 00000000 00001010

- 00000000 00000000 00000000 00010100

Reading and Writing Binary Data

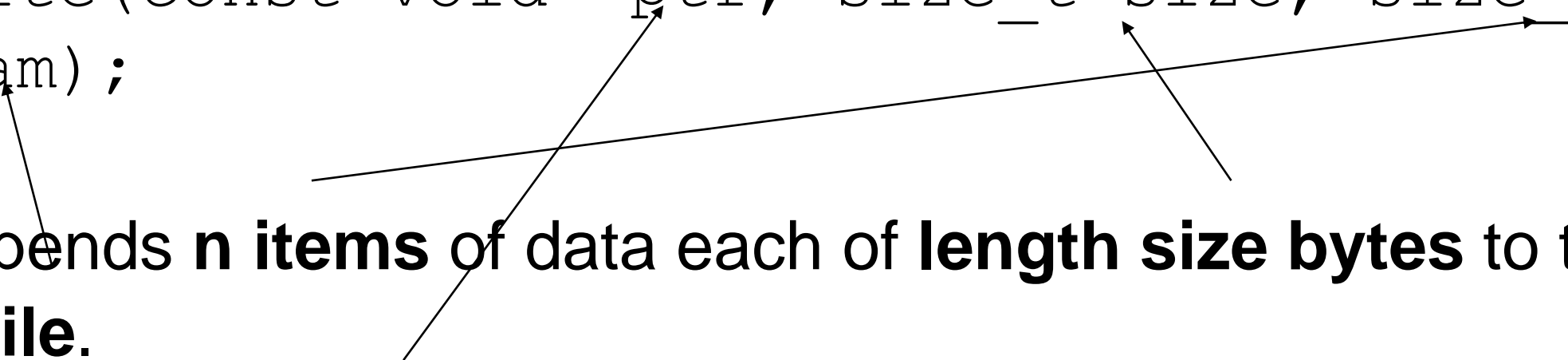
Reading in Binary:

```
FILE *f1Ptr = fopen("binary.dat", "rb");  
fread(&x, sizeof(int), 2, f1Ptr);
```

Writing in Binary:

```
FILE *f2Ptr = fopen("binary.dat", "wb");  
fwrite(&x, sizeof(int), 2, f2Ptr);
```

The `fwrite()` function

- `fwrite()` writes to a file in binary form
- `size_t fwrite(const void *ptr, size_t size, size_t n, FILE *stream);`
- `fwrite()` appends **n items** of data each of **length size bytes** to the given **output file**.
- The data written begins at **ptr**.
- The total number of bytes written is ($n * \text{size}$).
- `ptr` in the declarations is a pointer to any object.

For example:

```
fwrite(&myrec, sizeof(int), 6, fPtr);
```

- Size of each block is size of `int` (i.e. 4 byte)
- Write 6 blocks (i.e. 24 byte)
- Write to the current location of the file pointer pointed to by `f`.
- The block written is from the memory address `&myrec`.

Binary Records Example

Write the string "10 20"
(the bit stream: 00100001 00100000 00100000
00100010 00100000)

```
#include <stdio.h>
int main(void)
{ int x[2] = {10,20};
  FILE *f1Ptr = fopen("ascii.dat", "w");
  FILE *f2Ptr = fopen("binary.dat", "wb");
```

```
  fprintf(f1Ptr, "%d %d", x[0], x[1]);
  fwrite(&x, sizeof(int), 2, f2Ptr);
```

Write the binary of 10 and 20
(the bit stream: 00000000000001010
00000000000010100)

```
  fclose(f1Ptr);
  fclose(f2Ptr);
  return 0;
```

```
}
```

 ascii.dat - Notepad

File Edit Format View Help

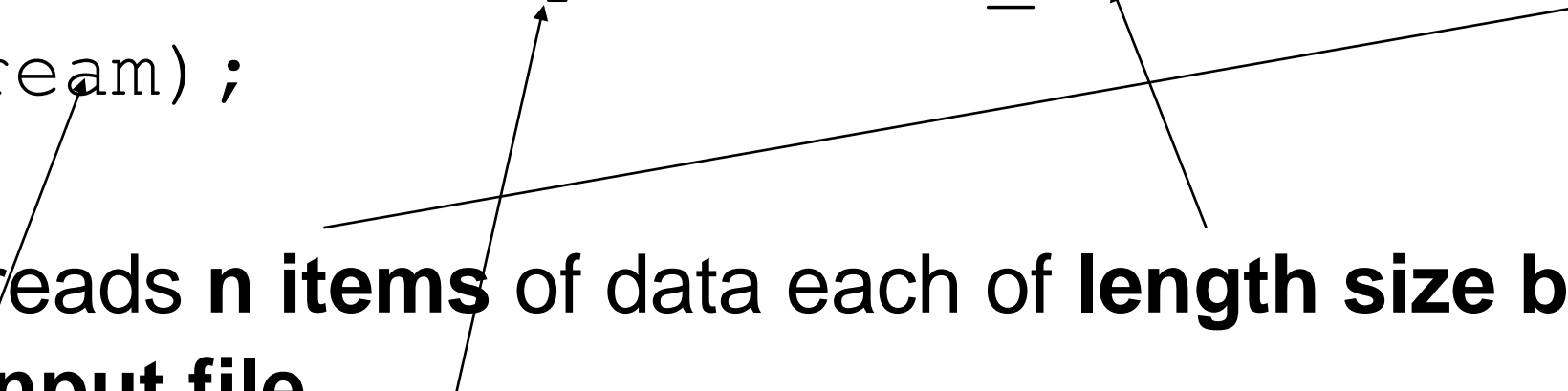
10 20

 binary.dat - Notepad

File Edit Format View Help

| 0

The `fread()` function

- `fread()` reads from a file in binary form
- `size_t fread(void *ptr, size_t size, size_t n, FILE *stream);`
- `fread()` reads **n items** of data each of **length size bytes** from the given **input file**.
- The data read begins at **ptr**.
- The total number of bytes written is $(n * \text{size})$.
- `ptr` in the declarations is a pointer to any object.

For example:

```
fread(&myrec, sizeof(int), 6, fPtr);
```

- Size of each block is size of `int` (i.e. 4 byte)
- Read 6 blocks (i.e. 24 byte)
- Read from the **current location** of the file pointer pointed to by `f`
- The block read is stored to the memory address `&myrec`.

Reading and Writing Binary Records

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

```
int main(void)
```

```
{ int x[2] = {10,20};
```

```
FILE *f1Ptr = fopen("binary.dat", "wb");
```

```
fwrite(&x, sizeof(int), 2, f1Ptr);
```

```
fclose(f1Ptr);
```

```
return 0;
```

```
}
```

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

```
int main(void)
```

```
{ FILE *f2Ptr = fopen("binary.dat", "rb");
```

```
int y[2] = {0};
```

```
fread(&y, sizeof(int), 2, f2Ptr);
```

```
printf("y: %d %d\n", y[0], y[1]);
```

```
fclose(f2Ptr);
```

```
return 0;
```

```
}
```



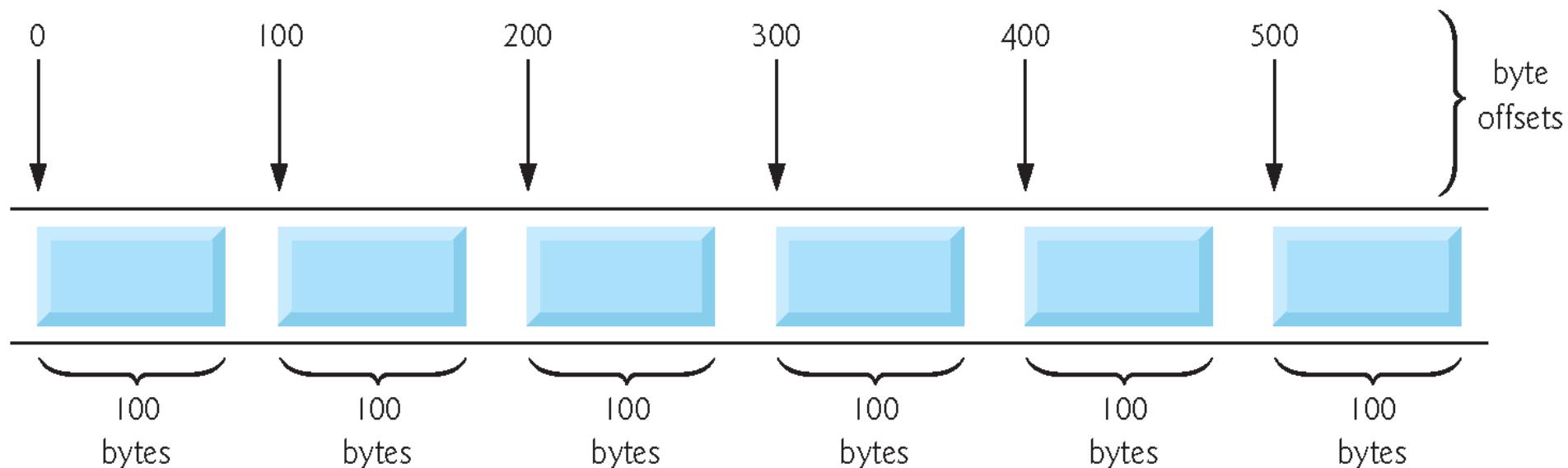
C:\Users\Arthur Tang\Documents\Untitled3.exe

y: 10 20

Process exited after 0.01396 seconds with return value 0
Press any key to continue . . .

Fixed-length Binary Records

- Fixed-length records enable data to be inserted in a random-access file *without destroying other data in the file*.
- Data stored previously can also be updated or deleted without rewriting the entire file.



Random Access File

- You can read from the beginning using the `fread()` function

```
fread(&y, sizeof(int), 9, f2Ptr);
```

- What if I need to read the five number (data in binary format)

- e.g.



69	72	84	89	68	95	99	98	76
----	----	----	----	----	----	----	----	----

- You can read the whole thing, and reference the 5th item
- But it is computationally consuming. Can I just read the 5th item from the file?
=> Random Access

Seek and Read

```
#include <stdio.h>
int main(void)
{ int x[5] = {10,20,30,40,50};
  FILE *f1Ptr = fopen("binary.dat", "wb");
  fwrite(&x, sizeof(int), 5, f1Ptr);
  fclose(f1Ptr);
  return 0;
}
```

```
#include <stdio.h>
int main(void)
{ FILE *f2Ptr = fopen("binary.dat", "rb");
  int y[2] = {0};
  int position = 2;
  fseek(f2Ptr, position*sizeof(int), 0);
  fread(&y, sizeof(int), 2, f2Ptr);
  printf("y: %d %d\n", y[0], y[1]);
  fclose(f2Ptr);
  return 0;
}
```

Move the current location to 2 x sizeof(int) [i.e. 8 byte], counting from the beginning of the file.

Read from the current position pointing to the file

 C:\Users\Arthur Tang\Documents\Untitled3.exe

y: 30 40

Process exited after 0.01604 seconds with return value 0
Press any key to continue . . .

fseek () Function

```
int fseek(FILE *stream, long offset, int whence);
```

Move the file pointer

- `fseek` sets the file pointer to a new position that is offset bytes from the file location given by `whence`.
- `whence` must be one of the values 0, 1, or 2 which represent three symbolic constants (defined in `stdio.h`) as follows:

Constant	Whence	File location
SEEK_SET	0	File beginning
SEEK_CUR	1	Current file pointer position
SEEK_END	2	End-of-file

fseek(), fread(), fwrite() Example

binary.dat is a file containing 10 integers in binary format, as follow:
11, 22, 33, 44, 55, 66, 77, 88, 99, 110

```
int x[3] = {0};  
FILE fPtr = fopen("binary.dat", "rb+");  
fseek(fPtr, 4*sizeof(int), 0);  
fread(&x, sizeof(int), 2, fPtr);  
fseek(fPtr, 2*sizeof(int), 1);  
fread(&x, sizeof(int), 1, fPtr);  
fseek(fPtr, -3*sizeof(int), 1);  
fread(&x, sizeof(int), 3, fPtr);  
fseek(fPtr, -6*sizeof(int), 2);  
fread(&x, sizeof(int), 3, fPtr);  
x[0]=1; x[1]=2; x[2]=3;  
fwrite(&x, sizeof(int), 3, fPtr);  
• printf("%d", ftell(fPtr));
```

Move the current pointer to the 4th integer counting from the beginning of the file (pointing to 55).

Read 2 integers from the file at the current pointer (55 and 66) and store it to the address of &x. Current pointer pointing to 77.

Move the current pointer to the +2 integers counting from the current pointer (99).

Read 1 integer from the file at the current pointer (99) and store it to the address of &x. Current pointer pointing to 110.

Move the current pointer to the -3 integers counting from the current pointer (77).

Read 3 integers from the file at the current pointer (77 88 99) and store it to the address of &x. Current pointer pointing to 110.

Move the current pointer to the -6 integers counting from the end of the file (55).

Write 3 integers to the file at the current pointer (position of 88 99 110) with the values stored at &x. binary.dat now becomes:
11, 22, 33, 44, 55, 66, 77, 1, 2, 3
Current position pointing to end-of-file (the 40th byte)

Read 3 integers from the file at the current pointer (55 66 77) and store it to the address of &x. Current pointer pointing to 88.

`ftell()` Function

```
int ftell(FILE *stream);
```

Return the current position of the file pointer (in terms of the n^{th} byte), counting from the beginning of the file.

Data Structure

- Before you read the data, you need to know if you are reading binary data or ascii data
- You also need to know what is the data structure/representation you're reading in
- Then you can design the logic for reading the data, and then the processing work afterward
- You can read/write/seek in various different data type size
 - e.g. char, int, float, double, etc...

Q&A?