“Character Input/Output and Input Validations”

Using Bloodshed Dev-C++

Heejin Park

Hanyang University

Introduction

- Single-Character I/O: `getchar()` and `putchar()`
- Buffers
- Terminating Keyboard Input
- Redirection and Files
- Creating a Friendlier User Interface
- Input Validation
- Menu Browsing

Single-Character I/O : getchar() and putchar()

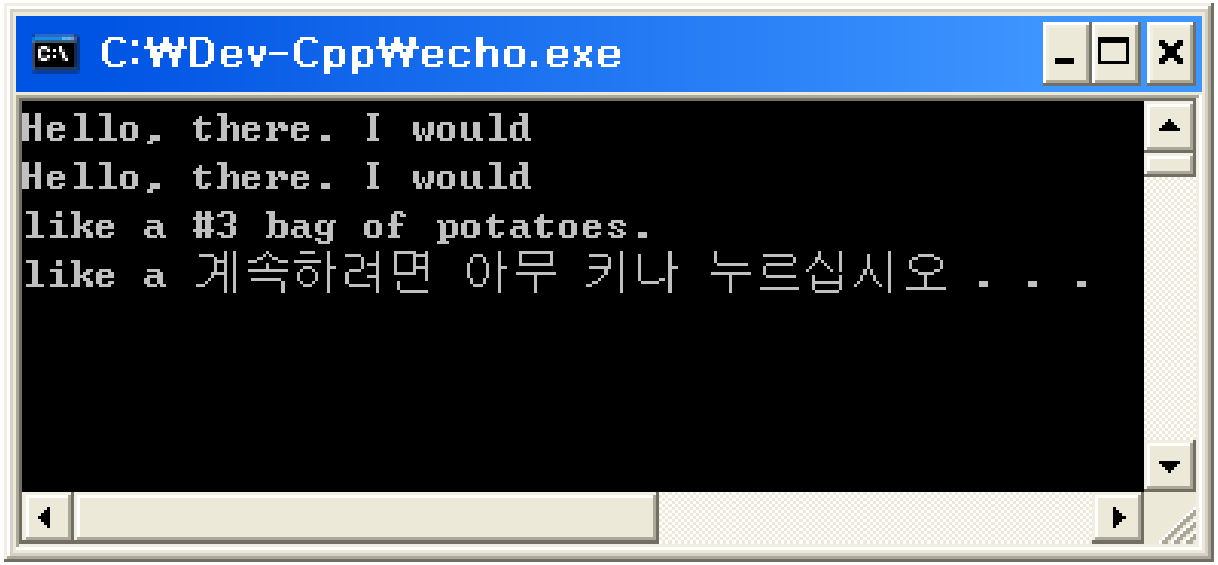
- The echo.c Program

```
#include <stdio.h>

int main(void)
{
    char ch;

    while ((ch = getchar()) != '#')
        putchar(ch);

    return 0;
}
```

Single-Character I/O : getchar() and putchar()

The echo.c Program

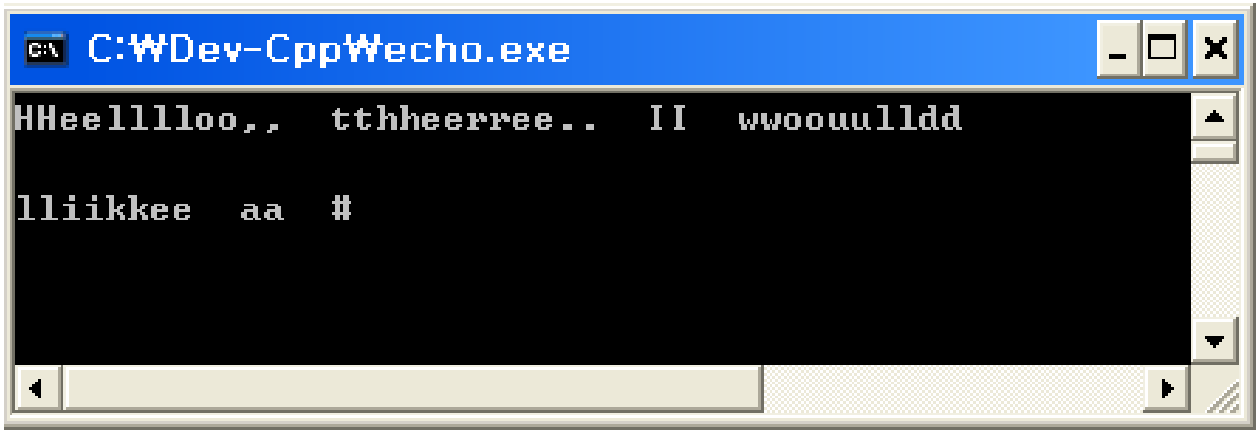
- ANSI C associates the `stdio.h` header file with using `getchar()` and `putchar()`, which is why we have included that file in the program.



Buffers

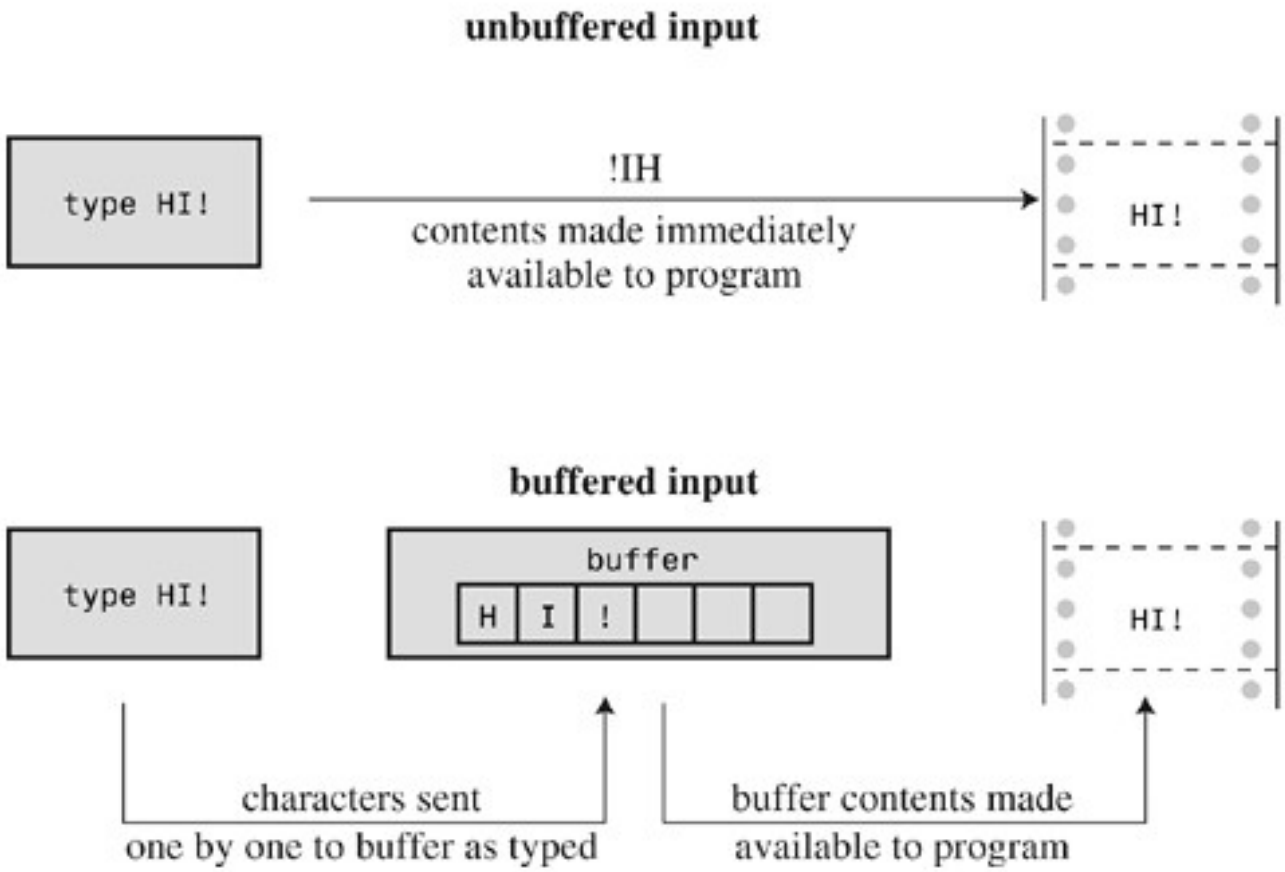
The echo.c Program

- When you run the previous program on some systems, the text you input is echoed immediately.
- That is, a sample run would look like this:



Buffers

- Buffered versus unbuffered input



Terminating Keyboard Input

The echo.c Program

- **Halts** when # is entered,
- which is convenient as long as you exclude that character from normal input.
- As you've seen, however, # can show up in normal input.

Terminating Keyboard Input

Files, Streams, and Keyboard Input

- **File**
- an area of memory in which information is stored.
- **Stream**
- an idealized flow of data to which the actual input or output is mapped.
- **Keyboard Input**
- represented by a stream called `stdin`.
- output to the screen is represented by a stream called `stdout`.

Terminating Keyboard Input

The End of File

- A computer operating system needs some way to tell where each file begins and ends.
- **How to detect end of file**
  - One method is to place a special character in the file to mark the end.
  - **A file with an end-of-file marker**

prose:

Ishphat the robot
slid open the hatch
and shouted his challenge.

prose in a file:

```
Ishphat the robot\nslid open the hatch\nand shouted his challenge.\n^Z
```

Terminating Keyboard Input

The End of File

- **How to detect end of file**
- A second approach is for the operating system to store information on the size of the file.
- **`getchar()` function**
  - return a special value when the end of a file is reached, regardless of how the operating system actually detects the end of file.

Terminating Keyboard Input

The End of File

- **`EOF`: end of file**
- The return value for `getchar()` when it detects an end of file is `EOF`.
- The `scanf()` function also returns `EOF` on detecting the end of a file.
- Typically, `EOF` is defined in the `stdio.h` file as follows:

```
#define EOF (-1)
```

  – Why -1?

Terminating Keyboard Input

The End of File

- You can use an expression like this:

```
while ((ch = getchar()) != EOF)
```

Terminating Keyboard Input

- **The echo_eof.c Program**

```c
#include <stdio.h>

int main(void)
{
    int ch;

    while ((ch = getchar()) != EOF)
        putchar(ch);

    return 0;
}
```

Terminating Keyboard Input

- The echo_eof.c Program

```
C:₩Dev-Cpp₩echo_eof.exe
She walks in beauty, like the night
She walks in beauty, like the night
  Of cloudless climes and starry skies...
  Of cloudless climes and starry skies...
                        Lord Byron
                        Lord Byron
```

Terminating Keyboard Input

The echo_eof.c Program

- **Note these points**
- You don't have to define `EOF` because **`stdio.h`** takes care of that.
- The `#define` statement in `stdio.h` enables you to use the symbolic representation `EOF`.
- The variable `ch` is changed from **type `char` to type `int`**.
  - because `char` variables may be represented by unsigned integers in the range `0` to `255`, but `EOF` may have the numeric value `-1`.

Terminating Keyboard Input

The echo_eof.c Program

- **Note these points**
- The fact that `ch` is an integer doesn't faze `putchar()`.
- To use this program on keyboard input,
  - you need a way to type the `EOF` character.
  - Ex) On most Unix systems
    - » pressing Ctrl+D at the beginning of a line causes the end-of-file signal to be transmitted.

Terminating Keyboard Input

The echo_eof.c Program

- Here is a buffered example of running echo_eof.c on a **Unix system**.

```
She walks in beauty, like the night
She walks in beauty, like the night
    Of cloudless climes and starry skies...
    Of cloudless climes and starry skies...
                                        Lord Byron
                                        Lord Byron
[Ctrl+D]
```

Terminating Keyboard Input

The echo_eof.c Program

- Here is a buffered example of running echo_eof.c on a PC.
- On a PC, you would press **Ctrl+Z** instead.

```
She walks in beauty, like the night
She walks in beauty, like the night
    Of cloudless climes and starry skies...
    Of cloudless climes and starry skies...
                                        Lord Byron
                                        Lord Byron
[Ctrl+Z]
```

Redirection and Files

Redirection and Files

- By default, a C program using the standard I/O package looks to the standard input as its source for input.
- This is the stream identified earlier as **`stdin`**.

- **Two ways to get a program to work with files.**
- One way is to explicitly use special functions.
  - open files, close files, read files, write in files, and so forth.
- The second way is to use a program designed to work with a keyboard and screen.

Redirection and Files

Unix, Linux, and DOS Redirection

- **Redirecting Input**
- Suppose
  - compiled the `echo_eof.c` program
  - placed the executable version in a file called `echo_eof`.
  - To run the program, type the executable file's name:

```
echo_eof
```

Redirection and Files

Unix, Linux, and DOS Redirection

- **Redirecting Input**
- Suppose
  - You want to use the program on a **text file** called `words`.

```
echo_eof < words
```

  - **The < symbol**
    - » a Unix and Linux (and DOS) redirection operator.
    - » It causes the `words` file to be associated with the `stdin` stream, channeling the file contents into the `echo_eof` program.

Redirection and Files

Unix, Linux, and DOS Redirection

- **Redirecting Input**
- Here is a sample run for one particular `words` file.
  - The `$` is one of the standard Unix and Linux prompts.

```
$ echo_eof < words
The world is too much with us: late and soon,
Getting and spending, we lay waste our powers:
Little we see in Nature that is ours;
We have given our hearts away, a sordid boon!
$
```

Redirection and Files

Unix, Linux, and DOS Redirection

- **Redirecting Output**
- Suppose
  - You want to have `echo_eof` send your keyboard input to a file called `mywords`.

```
echo_eof > mywords
```

  - The > is a second redirection operator.
    - » It causes a new file called `mywords` to be created for your use.
    - » Then it redirects the output of `echo_eof` to that file.

Redirection and Files

Unix, Linux, and DOS Redirection

- **Redirecting Output**
- To end the program, press Ctrl+D (Unix) or Ctrl+Z (DOS) at the beginning of a line.

```
$ echo_eof > mywords
You should have no problem recalling which redirection
operator does what. Just remember that each operator points
in the direction the information flows. Think of it as
a funnel.
[Ctrl+D]
$
```

Redirection and Files

Unix, Linux, and DOS Redirection

- **Redirecting Output**
- You can use the Unix and Linux `cat` or DOS `type` command to check the contents, or you can use `echo_eof` again.
- this time redirecting the file to the program:

```
$ echo_eof < mywords
You should have no problem recalling which redirection
operator does what. Just remember that each operator points
in the direction the information flows. Think of it as a
funnel.
$
```

Redirection and Files

Unix, Linux, and DOS Redirection

- **Combined Redirection**
- Suppose
  - You want to make a copy of the file `mywords` and call it `savewords`.

```
echo_eof < mywords > savewords
```

  - The following command would have worked as well.
    - » because the order of redirection operations doesn't matter:

```
echo_eof > savewords < mywords
```

Redirection and Files

Unix, Linux, and DOS Redirection

- **Combined Redirection**
- Beware
  - Don't use the same file for both input and output to the same command.

```
echo_eof < mywords > mywords....<--WRONG
```

  - The reason is that `> mywords` causes the original `mywords` to be truncated to zero length before it is ever used as input.

Redirection and Files

Unix, Linux, and DOS Redirection

- **Rules**
- 1) A redirection operator connects an **executable** program with a data file.
- 2) Input cannot be taken from more than one file,

  nor can output be directed to more than one file.
- 3) Normally, spaces between the names and operators are optional.

Redirection and Files

Unix, Linux, and DOS Redirection

- **Some wrong examples**
- With `addup` and `count` as executable programs and `fish` and `beets` as text files:

`fish > beets`	**Violates the first rule**
`addup < count`	**Violates the first rule**
`addup < fish < beets`	**Violates the second rule**
`count > beets fish`	**Violates the second rule**

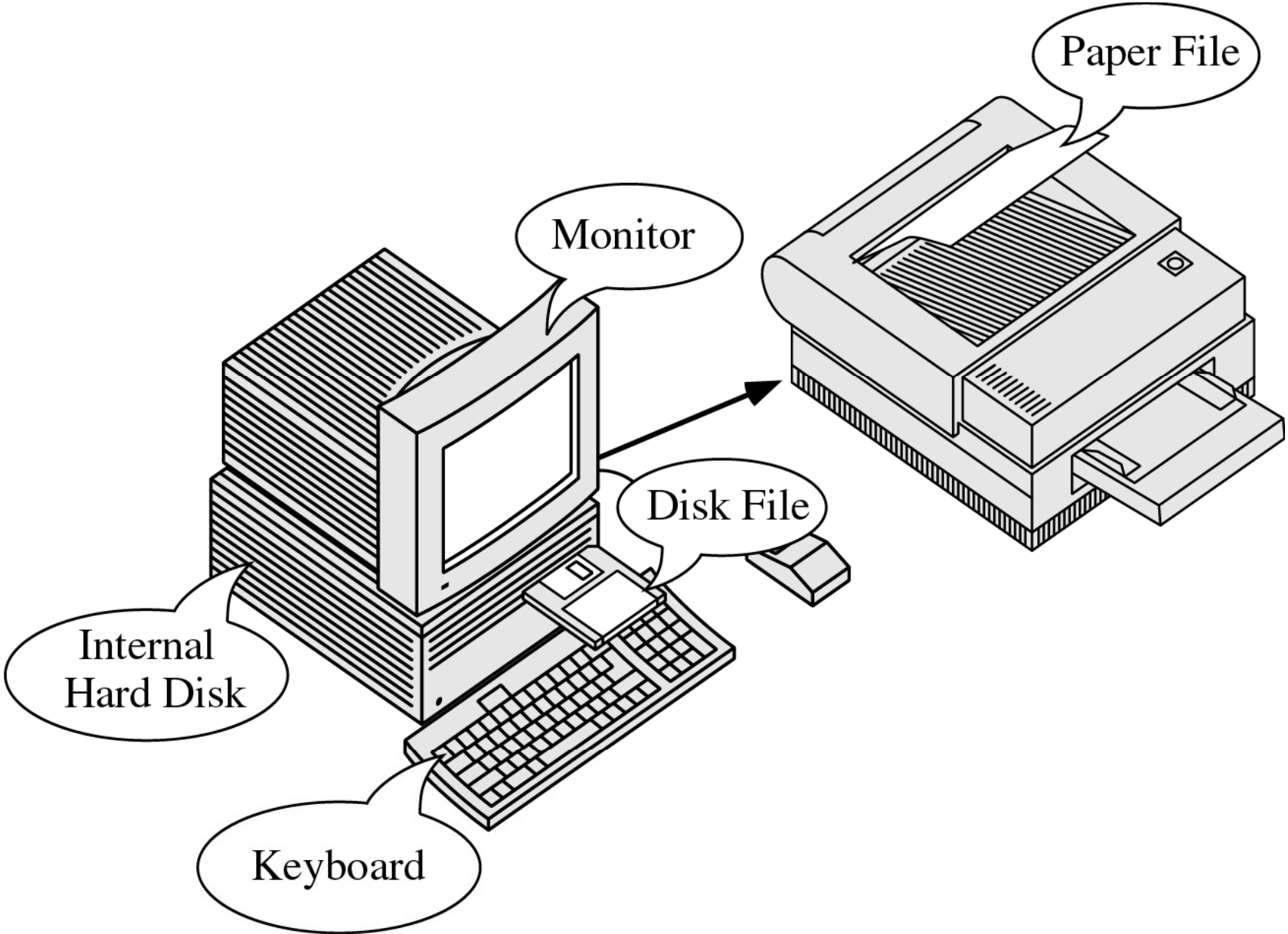
Redirection and Files

Unix, Linux, and DOS Redirection

- **Some wrong examples**
- With `addup` and `count` as executable programs and `fish` and `beets` as text files:

`fish > beets`	**Violates the first rule**
`addup < count`	**Violates the first rule**
`addup < fish < beets`	**Violates the second rule**
`count > beets fish`	**Violates the second rule**

Files in a personal computer environment

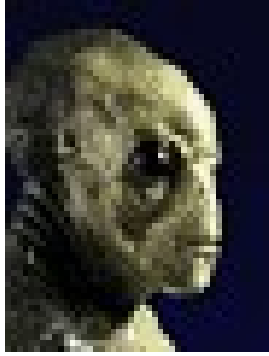


Using Input/Output Files

- ***stream*** - a sequence of characters
  - interactive
    - **stdin** - input stream associated with **keyboard.**
    - **stdout** - output stream associated with **display.**
  - file

```c
#include <stdio.h>

int main()
{
    int ch;
    while((ch=fgetc(stdin))!=EOF)
        fputc(ch, stdout);
    return 0;

}
```

Using Input/Output Files



- ***stream*** - a sequence of characters
  - interactive
    - **stdio** - input stream associated with **keyboard.**
    - **stdout** - output stream associated with **display.**
  - file stream

```
int main()
{
    int ch;
    FILE *fp_in, *fp_out;
    ...

    while((ch=fgetc(fp_in))!=EOF)
        fputc(ch, fp_out);
    return 0;
}
```

File Open

- The file open function (**`fopen`**) makes the connection between the physical file and the stream.
- Syntax:
  `fopen("filename", "mode");`

File Open

- The file open function (**`fopen`**) makes the connection between the physical file and the stream.
- Syntax:
  `fopen("filename", "mode");`
- ***`mode`*** tells C how the program will use the file.
- We assign the return value of **`fopen`** to our pointer variable:

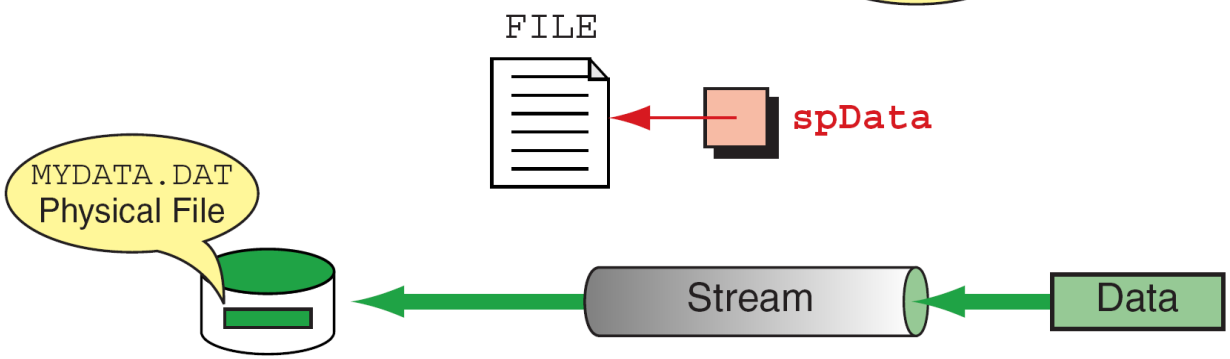
```
spData = fopen("MYFILE.DAT", "w");
spData = fopen("A:\\MYFILE.DAT", "w");
spData = fopen("/home/st/MYFILE.DAT", "w");
```

More On `fopen`

```
#include <stdio.h>
…
{
int main (void)
  FILE* spData;
  …
  spData = fopen("MYDATA.DAT", "w");
  …
} // main
```

Internal File Variable

External File Name



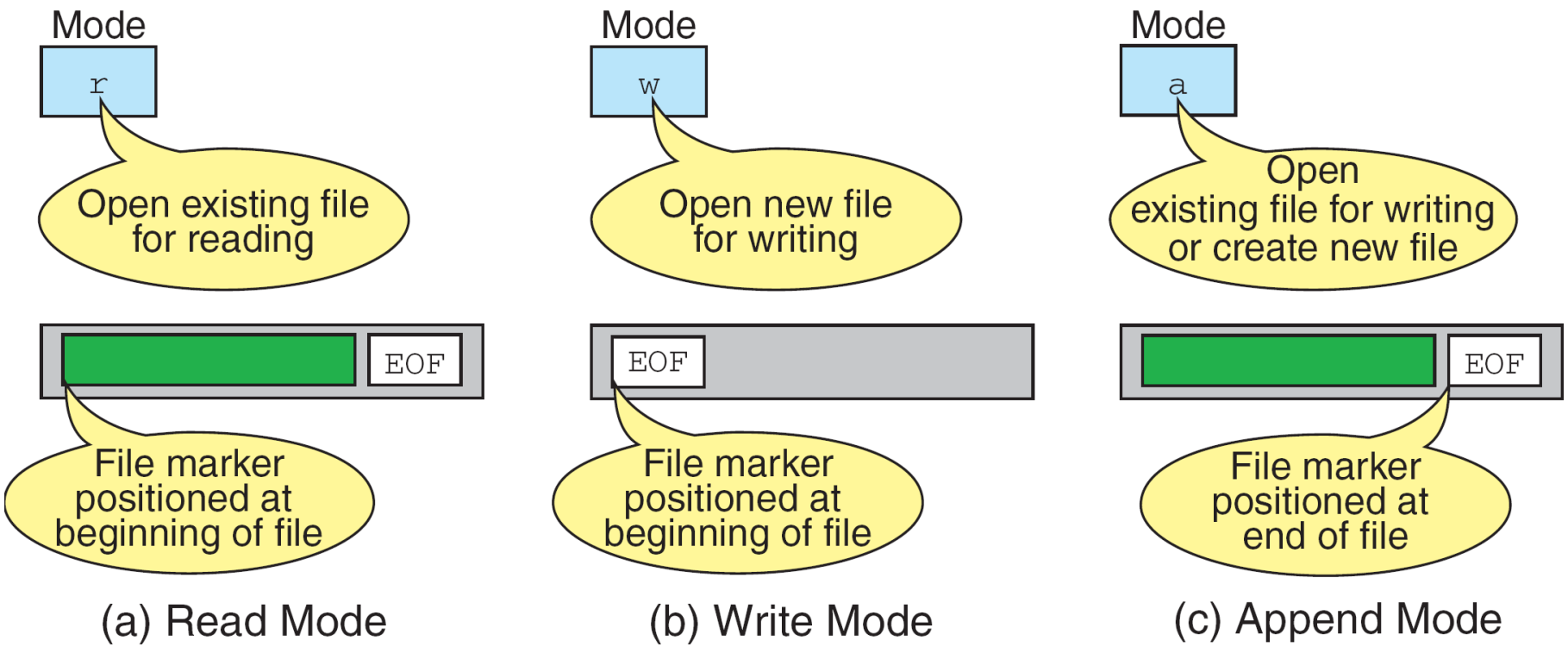
from Figure 7-3 in* Forouzan & Gilberg, *p. 399

File Open Modes

Mode	Meaning
r	Open text file in read mode • If file exists, the marker is positioned at beginning. • If file doesn't exist, error returned.
w	Open text file in write mode • If file exists, it is erased. • If file doesn't exist, it is created.
a	Open text file in append mode • If file exists, the marker is positioned at end. • If file doesn't exist, it is created.

from Table 7-1 in* Forouzan & Gilberg*, p. 400

More on File Open Modes



from Figure 7-4 in* Forouzan & Gilberg*, p. 401

Closing a File

- When we finish with a mode, we need to close the file before ending the program or beginning another mode with that same file.
- To close a file, we use **`fclose`** and the pointer variable:

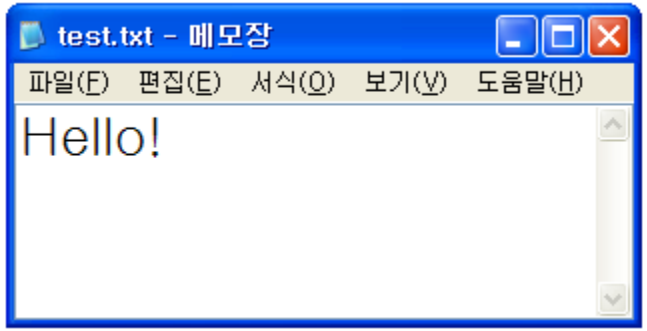
```
fclose(spData);
```

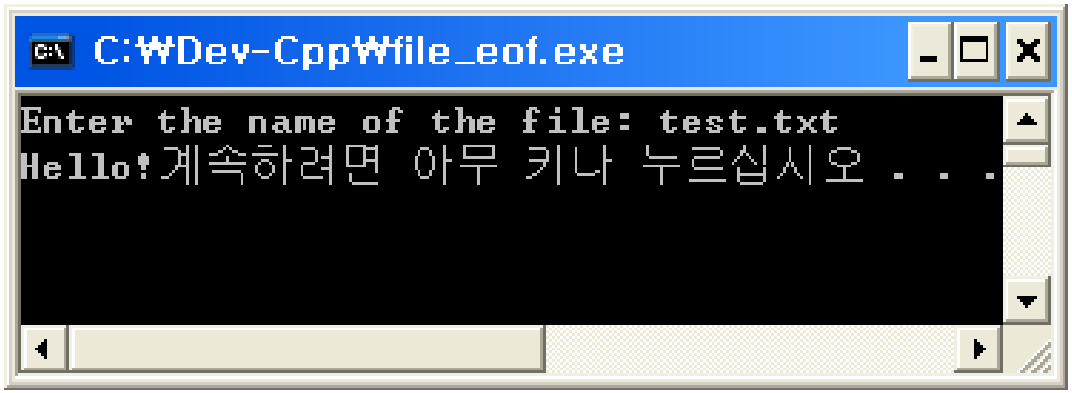
file_eof.c

```c
#include <stdio.h>
#include <stdlib.h>  // for exit()

int main()
{
    int ch;
    FILE * fp;
    char fname[50];          // to hold the file name

    printf("Enter the name of the file: ");
    scanf("%s", fname);
    fp = fopen(fname, "r"); // open file for reading

    if (fp == NULL)         // attempt failed
    {
       printf("Failed to open file. Bye\n");
       exit(1);             // quit program
    }
// getc(fp) gets a character from the open file

    while ((ch = getc(fp)) != EOF)
        putchar(ch);
    fclose(fp);             // close the file

    return 0;
}
```

Redirection and Files

The file_eof.c Program





Additional I/O Functions

Terminal Input/Output

```
scanf ("control string", ...);
printf("control string", ...);
```

General Input/Output

```
fscanf (stream_pointer, "control string", ...);
fprintf(stream_pointer, "control string", ...);
```

from Table 7-2 in* Forouzan & Gilberg*, p. 403

Creating a Friendlier User Interface

The guess.c Program

- **Working with Buffered Input**

```
#include <stdio.h>

int main(void)
{
    int guess = 1;

    printf("Pick an integer from 1 to 100. I will try to guess ");
    printf("it.\nRespond with a y if my guess is right and with");
    printf("\nan n if it is wrong.\n");
    printf("Uh...is your number %d?\n", guess);

    while (getchar() != 'y')      /* get response, compare to y */
        printf("Well, then, is it %d?\n", ++guess);
    printf("I knew I could do it!\n");

    return 0;
}
```

Creating a Friendlier User Interface

The guess.c Program

```
C:₩Dev-Cpp₩guess.exe
Pick an integer from 1 to 100. I will try to guess it.
Respond with a y if my guess is right and with
an n if it is wrong.
Uh...is your number 1?
n
Well, then, is it 2?
Well, then, is it 3?
n
Well, then, is it 4?
Well, then, is it 5?
y
I knew I could do it!
계속하려면 아무 키나 누르십시오 . . .
```

Creating a Friendlier User Interface

The guess.c Program

- What's happening is that the program reads the `n` response as a denial that the number is 1 and then reads the newline character as a denial that the number is 2.

Creating a Friendlier User Interface

The guess.c Program

- One solution is to use a `while` loop to discard the rest of the input line, including the newline character.

```
while (getchar() != 'y')    /* get response, compare to y */
{
    printf("Well, then, is it %d?\n", ++guess);

    while (getchar() != '\n')
        continue;           /* skip rest of input line    */
}
```

Creating a Friendlier User Interface

The guess.c Program

- Using this loop produces responses such as the following:

```
C:₩Dev-Cpp₩guess_2.exe
Pick an integer from 1 to 100. I will try to guess it.
Respond with a y if my guess is right and with
an n if it is wrong.
Uh...is your number 1?
n
Well, then, is it 2?
no
Well, then, is it 3?
no sir
Well, then, is it 4?
forget it
Well, then, is it 5?
y
I knew I could do it!
계속하려면 아무 키나 누르십시오 . . .
```

Creating a Friendlier User Interface

The guess.c Program

- You might not like `f` being treated the same as `n`.
- To eliminate that defect, you can use **an `if` statement** to screen out other responses.
- First, **add a `char` variable** to store the response:

```
char response;
```

Creating a Friendlier User Interface

The guess.c Program

- Then change the loop to this:

```
while ((response = getchar()) != 'y')      /* get response */
{
   if (response == 'n')
      printf("Well, then, is it %d?\n", ++guess);
   else
      printf("Sorry, I understand only y or n.\n");

   while (getchar() != '\n')
      continue;                    /* skip rest of input line */
}
```

Creating a Friendlier User Interface

The guess.c Program

- Now the program's response looks like this:

```
C:₩Dev-Cpp₩guess_2.exe
Pick an integer from 1 to 100. I will try to guess it.
Respond with a y if my guess is right and with
an n if it is wrong.
Uh...is your number 1?
n
Well, then, is it 2?
no
Well, then, is it 3?
no sir
Well, then, is it 4?
forget it
Sorry, I understand only y or n.
n
Well, then, is it 5?
y
I knew I could do it!
계속하려면 아무 키나 누르십시오 . . .
```

Creating a Friendlier User Interface

- The showchar1.c Program using getchar

```
C:₩Dev-Cpp₩showchar1.exe
Enter a character and two integers:
c 2 3
ccc
ccc
Enter another character and two integers;
Enter a newline to quit.
Bye.
계속하려면 아무 키나 누르십시오 . . .
```

Creating a Friendlier User Interface

The showchar1.c Program(1/2)

```
#include <stdio.h>

void display(char cr, int lines, int width);
int main(void)
{
    int ch;                 /* character to be printed    */
    int rows, cols;         /* number of rows and columns */
    printf("Enter a character and two integers:\n");

    while ((ch = getchar()) != '\n')
    {
        scanf("%d %d", &rows, &cols);
        display(ch, rows, cols);
        printf("Enter another character and two integers;\n");
        printf("Enter a newline to quit.\n");
    }
    printf("Bye.\n");
    return 0;
}
```

Creating a Friendlier User Interface

The showchar1.c Program(2/2)

```
void display(char cr, int lines, int width)
{
    int row, col;

    for (row = 1; row <= lines; row++)
    {
        for (col = 1; col <= width; col++)
            putchar(cr);
        putchar('\n');  /* end line and start a new one */
    }
}
```

Creating a Friendlier User Interface

The showchar2.c Program

```
C:₩Dev-Cpp₩showchar2.exe
Enter a character and two integers:
c 1 2
cc
Enter another character and two integers;
Enter a newline to quit.
! 3 6
!!!!!!
!!!!!!
!!!!!!
Enter another character and two integers;
Enter a newline to quit.

Bye.
계속하려면 아무 키나 누르십시오 . . .
```

Creating a Friendlier User Interface

The showchar2.c Program(1/2)

```
#include <stdio.h>

void display(char cr, int lines, int width);

int main(void)
{
    int ch;             /* character to be printed      */
    int rows, cols;     /* number of rows and columns   */

    printf("Enter a character and two integers:\n");

    while ((ch = getchar()) != '\n')
    {
        if (scanf("%d %d",&rows, &cols) != 2)
            break;
        display(ch, rows, cols);

        while (getchar() !=  '\n')
            continue;

        printf("Enter another character and two integers;\n");
        printf("Enter a newline to quit.\n");
    }
    printf("Bye.\n");
    return 0;
}
```

Creating a Friendlier User Interface

The showchar2.c Program(2/2)

```
void display(char cr, int lines, int width)
{
    int row, col;

    for (row = 1; row <= lines; row++)
    {
        for (col = 1; col <= width; col++)
            putchar(cr);
        putchar('\n');  /* end line and start a new one */
    }
}
```

Menu Browsing

- The menuette.c

C:₩Dev-Cpp₩menuette.exe

```
Enter the letter of your choice:
a. advice           b. bell
c. count            q. quit
a
Buy low, sell high.
Enter the letter of your choice:
a. advice           b. bell
c. count            q. quit
count
Count how far? Enter an integer:
two
two is not an integer.
Please enter an integer value, such as 25, -178, or 3: 5
1
2
3
4
5
Enter the letter of your choice:
a. advice           b. bell
c. count            q. quit
d
Please respond with a, b, c, or q.
q
Bye.
계속하려면 아무 키나 누르십시오 . . .
```

Input Validation

Input Validation

- **Suppose**
- For instance, that you had a loop that processes nonnegative numbers.
- One kind of error the user can make is to enter a negative number.
  - You can use a relational expression to test for that:

```
int n;
scanf("%d", &n);        // get first value

while (n >= 0)              // detect out-of-range value
{
    // process n
    scanf("%d", &n);   // get next value
}
```

Input Validation

Input Validation

- Another potential pitfall
- The user might enter the wrong type of value, such as the character `q`.
- One way to detect this kind of misuse
  - to check the return value of `scanf()`.

```
scanf("%d", &n) == 1
```

Input Validation

Input Validation

- This suggests the following revision of the code:

```
int n;

while (scanf("%d", &n) == 1 && n >= 0)
{
    // process n
}
```

- "while input is an integer and the integer is positive."

Input Validation

Input Validation

- Here the fact that input really is a stream of characters comes in handy.
- because you can use `getchar()` to read the input character-by-character.

```
int get_int(void)
{
    int input;
    char ch;

    while (scanf("%d", &input) != 1)
    {
        while ((ch = getchar()) != '\n')
            putchar(ch);  // dispose of bad input
        printf(" is not an integer.\nPlease enter an ");
        printf("integer value, such as 25, -178, or 3: ");
    }
    return input;
}
```

Input Validation

The stdbool.h header file

- If you don't have `_Bool` on your system,
- you can substitute `int` for `bool`, `1` for `true`, and `0` for `false`.
- Note that the function returns `true` if the input is invalid.
  - Hence the name `bad_limits()`:

Input Validation

- The stdbool.h header file

```
bool bad_limits(int begin, int end, int low, int high)
{
    bool not_good = false;
    if (begin > end)
    {
        printf("%d isn't smaller than %d.\n", begin, end);
        not_good = true;
    }
    if (begin < low || end < low)
    {
        printf("Values must be %d or greater.\n", low);
        not_good = true;
    }
    if (begin > high || end > high)
    {
        printf("Values must be %d or less.\n", high);
        not_good = true;
    }
        return not_good;
}
```

Input Validation

The checking.c Program(1/4)

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

int get_int(void);
bool bad_limits(int begin, int end, int low, int high);
double sum_squares(int a, int b);

int main(void)
{
    const int MIN = -1000;  // lower limit to range
    const int MAX = +1000;  // upper limit to range
    int start;              // start of range
    int stop;               // end of range
    double answer;

    printf("This program computes the sum of the squares of "
           "integers in a range.\nThe lower bound should not "
           "be less than -1000 and\nthe upper bound should not "
           "be more than +1000.\nEnter the limits (enter 0 for "
           "both limits to quit):\nlower limit: ");
    start = get_int();
    printf("upper limit: ");
    stop = get_int();
```

Input Validation

The checking.c Program(2/4)

```
    while (start !=0 || stop != 0)
    {
        if (bad_limits(start, stop, MIN, MAX))
            printf("Please try again.\n");
        else
        {
            answer = sum_squares(start, stop);
            printf("The sum of the squares of the integers ");
            printf("from %d to %d is %g\n", start, stop, answer);
        }
        printf("Enter the limits (enter 0 for both "
               "limits to quit):\n");
        printf("lower limit: ");

        start = get_int();
        printf("upper limit: ");
        stop = get_int();
    }
    printf("Done.\n");
    return 0;
}
```

Input Validation

The checking.c Program(3/4)

```
int get_int(void)
{
    int input;
    char ch;

    while (scanf("%d", &input) != 1)
    {
        while ((ch = getchar()) != '\n')
            putchar(ch);  // dispose of bad input
        printf(" is not an integer.\nPlease enter an ");
        printf("integer value, such as 25, -178, or 3: ");
    }
    return input;
}

double sum_squares(int a, int b)
{
    double total = 0;
    int i;
    for (i = a; i <= b; i++)
        total += i * i;
    return total;
}
```

Input Validation

The checking.c Program(4/4)

```
bool bad_limits(int begin, int end, int low, int high)
{
    bool not_good = false;

    if (begin > end)
    {
        printf("%d isn't smaller than %d.\n", begin, end);
        not_good = true;
    }
    if (begin < low || end < low)
    {
        printf("Values must be %d or greater.\n", low);
        not_good = true;
    }
    if (begin > high || end > high)
    {
        printf("Values must be %d or less.\n", high);
        not_good = true;
    }
    return not_good;
}
```

Input Validation

The checking.c Program

```
C:₩Dev-Cpp₩checking.exe
This program computes the sum of the squares of integers in a range.
The lower bound should not be less than -1000 and
the upper bound should not be more than +1000.
Enter the limits (enter 0 for both limits to quit):
lower limit: low
low is not an integer.
Please enter an integer value, such as 25, -178, or 3: 3
upper limit: a big number
a big number is not an integer.
Please enter an integer value, such as 25, -178, or 3: 12
The sum of the squares of the integers from 3 to 12 is 645
Enter the limits (enter 0 for both limits to quit):
lower limit: 80
upper limit: 10
80 isn't smaller than 10.
Please try again.
Enter the limits (enter 0 for both limits to quit):
lower limit: 0
upper limit: 0
Done.
계속하려면 아무 키나 누르십시오 . . .
```

Input Validation

The checking.c Program

- **Analyzing the Program**
- The computational core (the function `sum_squares()`) is short.
  - but the input validation support makes it more involved.

- **The `main()` function**
  - It uses `get_int()` to obtain values
  - a `while` loop to process them
  - the `badlimits()` function to check for valid values
  - the `sum_squares()` function to do the actual calculation.

Input Validation

The checking.c Program

- The `main()` function

```
    start = get_int();
    printf("upper limit: ");
    stop = get_int();

    while (start !=0 || stop != 0)
    {
        if (bad_limits(start, stop, MIN, MAX))
            printf("Please try again.\n");
        else
        {
            answer = sum_squares(start, stop);
            printf("The sum of the squares of the integers ");
            printf("from %d to %d is %g\n", start, stop, answer);
        }
        printf("Enter the limits (enter 0 for both "
               "limits to quit):\n");
        printf("lower limit: ");

        start = get_int();
        printf("upper limit: ");
        stop = get_int();
    }
```

Input Validation

The checking.c Program

- **The Input Stream and Numbers**
- Consider a line of input like the following:

```
is  28 12.4
```

  - To a C program it looks like a stream of bytes.
  - i → s → space character → 2 → and so on.

Input Validation

The checking.c Program

- **The Input Stream and Numbers**
- So if `get_int()` encounters this line, the following code reads and discards the entire line, including the numbers, which just are other characters on the line:

```
while ((ch = getchar()) != '\n')
    putchar(ch);  // dispose of bad input
```

Input Validation

The checking.c Program

- **The Input Stream and Numbers**
- Although the input stream consists of characters
  - The `scanf()` function can convert them to a numeric value if you tell it to.

```
42
```

Menu Browsing

Menu Browsing

- Many computer programs use menus as part of the user interface.
- A menu offers the user a choice of responses.

```
Enter the letter of your choice:

a. advice                b. bell
c. count                 q. quit
```

Menu Browsing

Tasks

- You can use a while statement to provide repeated access to the menu.

```
get choice
while choice is not 'q'
    switch to desired choice and execute it
    get next choice
```

Menu Browsing

Toward a Smoother Execution

- Combining that with a `while` loop and a `switch`.

```
#include <stdio.h>

char get_choice(void);
void count(void);

int main(void)
{
    int choice;

    while ( (choice = get_choice()) != 'q')
    {
        switch (choice)
        {
            case 'a' :  printf("Buy low, sell high.\n");
                        break;
            case 'b' :  putchar('\a');  /* ANSI */
                        break;
            case 'c' :  count();
                        break;
            default  :  printf("Program error!\n");
                        break;
        }
    }
    return 0;
}
```

Menu Browsing

The get_choice() Function

- Here, in pseudocode, is one possible design for this function:

```
show choices
get response
while response is not acceptable
    prompt for more response
    get response
```

Menu Browsing

The get_choice() Function

- And here is a simple, but awkward, implementation:

```
char get_choice(void)
{
    int ch;

    printf("Enter the letter of your choice:\n");
    printf("a. advice            b. bell\n");
    printf("c. count             q. quit\n");
    ch = getchar();

    while (  (ch < 'a' || ch > 'c') && ch != 'q')
    {
        printf("Please respond with a, b, c, or q.\n");
        ch = getchar();
    }
    return ch;
}
```

Menu Browsing

The get_choice() Function

- You can rewrite the input function as follows:

```
char get_choice(void)
{
    int ch;

    printf("Enter the letter of your choice:\n");
    printf("a. advice           b. bell\n");
    printf("c. count            q. quit\n");
    ch = get_first();

    while (  (ch < 'a' || ch > 'c') && ch != 'q')
    {
        printf("Please respond with a, b, c, or q.\n");
        ch = getfirst();
    }
    return ch;
}
char get_first(void)
{
    int ch;
    ch = getchar();            /* read next character */

    while (getchar() != '\n')
        continue;              /* skip rest of line */
    return ch;
}
```

Menu Browsing

Mixing Character and Numeric Input

- Suppose
- Ex) the `count()` function (choice `c`) were to look like this:

```
void count(void)
{
    int n,i;

    printf("Count how far? Enter an integer:\n");
    scanf("%d", &n);

    for (i = 1; i <= n; i++)
        printf("%d\n", i);
}
```

Menu Browsing

Mixing Character and Numeric Input

- If you then responded by entering `3`,
- `scanf()` would read the 3 and leave a newline character as the next character in the input queue.
- The next call to `get_choice()` would result in `get_first()` returning this newline character, leading to undesirable behavior.
  - One way to fix that problem is to **rewrite `get_first()`**.

Menu Browsing

Mixing Character and Numeric Input

- A second approach is have the `count()` function tidy up and clear the newline itself.

```
void count(void)
{
    int n,i;

    printf("Count how far? Enter an integer:\n");
    n = get_int();

    for (i = 1; i <= n; i++)
        printf("%d\n", i);
    while ( getchar() != '\n')
        continue;
}
```

Menu Browsing

The menuette.c Program(1/3)

```
#include <stdio.h>

char get_choice(void);
char get_first(void);
int get_int(void);
void count(void);
int main(void)
{
    int choice;
    void count(void);
    while ( (choice = get_choice()) != 'q')
    {
        switch (choice)
        {
            case 'a' :  printf("Buy low, sell high.\n");
                        break;
            case 'b' :  putchar('\a');  /* ANSI */
                        break;
            case 'c' :  count();
                        break;
            default  :  printf("Program error!\n");
                        break;
        }
    }
    printf("Bye.\n");
    return 0;
}
```

Menu Browsing

The menuette.c Program(2/3)

```
void count(void)
{
    int n,i;

    printf("Count how far? Enter an integer:\n");
    n = get_int();
    for (i = 1; i <= n; i++)
        printf("%d\n", i);
    while ( getchar() != '\n')
        continue;
}

char get_choice(void)
{
    int ch;
    printf("Enter the letter of your choice:\n");
    printf("a. advice           b. bell\n");
    printf("c. count            q. quit\n");
    ch = get_first();

    while (  (ch < 'a' || ch > 'c') && ch != 'q')
    {
        printf("Please respond with a, b, c, or q.\n");
        ch = get_first();
    }
    return ch;
}
```

Menu Browsing

The menuette.c Program(3/3)

```
char get_first(void)
{
    int ch;
    ch = getchar();

    while (getchar() != '\n')
        continue;
    return ch;
}

int get_int(void)
{
    int input;
    char ch;

    while (scanf("%d", &input) != 1)
    {
        while ((ch = getchar()) != '\n')
            putchar(ch);  // dispose of bad input
        printf(" is not an integer.\nPlease enter an ");
        printf("integer value, such as 25, -178, or 3: ");
    }
    return input;
}
```