C Programming

Getting Input

Getting Input From The Keyboard

Prelude to The Examples

Assume that all of the following variable declarations apply to the examples:

```
char str1[81] = "", str2[81] = "", str3[81] = "";
int i = 0, j = 0;
float f = 0;
double d = 0;
```

Getting Keyboard Input

There are several ways that you can use to get keyboard input in C.

Some are better than others.

Getting A Single String From The Keyboard

gets() can be used to get a string from the keyboard

It takes all the input you enter until you press ENTER and puts it into an array of char that you specify.

e.g. gets(str1);

Problem With gets()

gets() does not check for overflowing the array you specify.

This is dangerous!

Review!

Why is gets() dangerous?

A Much Better, Slightly More Complicated Alternative

fgets() is another function that you can use that will get a string from a file, while specifying a maximum size for the string.

We can treat the keyboard as a file!

The keyboard has a special system variable set up for it: stdin.

We can use that variable with file-related functions.

Use of fgets()

fgets() takes three parameters:

1. the array of char in which you will put the string
2. the number of characters you are willing to accept, including the null-termination
3. the file from which you want to get the string (which would always be stdin if you want to get the string from the keyboard)

Review!

Why is fgets() less dangerous than gets()?

Example of fgets()

e.g. fgets(str1, 81, stdin);

This will get up to 80 (not 81) characters from the keyboard and put them in str1.

If the user enters
less than 80 characters
before pressing ENTER,
the string entered will get put into
str1 **and** a carriage return (‘\n’) will
be put at the end of the string,
before the null-termination.

If the user enters
80 characters or more,
the carriage return is left out
but
the null-termination is put in.

This allows you to tell if the user has entered more characters than you will accept into the string.

Major Difference with gets()

gets() will not put in the '\n'.
fgets() will.

Remember this!

gets() vs. fgets()

Due to the overflow protection from fgets(), always use fgets() with the stdin variable as the last parameter, instead of using gets().

Review!

What function is recommended for string input?

What If You Want To Get Something Besides One String?

There are other options for formatted input.

scanf()

scanf() is a function that is heavily used in textbooks but is mostly avoided in real world situations

scanf() gets formatted input from the keyboard

It uses a format string similar to that found in printf()

- scanf() scans the input string gotten from the keyboard, looking for the items specified in the format string
    - By default, the items are separated by spaces

Examples of Format Strings

Format String	Meaning
"%d %d %d"	Look for three consecutive integers, separated by spaces.
"%d %f %d"	Look for an integer followed by a floating point number, followed by an integer.
%s %d	Look for a string followed by an integer.
%d %s %d	Look for an integer, followed by a string, followed by an integer.

So, What's a String to scanf()?

scanf() considers a string to be any sequence of characters terminated by whitespace or the end of the line

Thus, in user input, an integer can also be a string!

Also, if you have two words as input, the space between them will separate them into two strings!

Examples of Matches to Format Strings

Format String	Sample Match
"%d %d %d"	170 200 201
"%d %f %d"	170 20.3 10
%s %d	fred 20
%s %d	10 20
%s %d %s	10 20 fred
%d %s %d	20 fred 40
%s %s %s	fred twinkletoes flintstone

So, What If You Want More Than One Word In A String?

We can use a “scanset”.

A scanset is a formatting code that is specified by %[], where the characters we want to accept as part of the string are between the square brackets.

e.g. "%[ABCD]" would accept a string as long as it had one of A, B, C, or D as one of the characters

Scanset Example

If we accepted "%[ABCD]" as our string, then any of the following would be OK:

- "AB"
- "ABABCDA"
- "AAAA"
- "D"

Terminating a scanset Input

When using a scanset, the string ends as soon as a character that is NOT in the scanset is found:

- e.g. scanset “%[ABCD]”: input string “ABABDPAB” would match “ABABD” because the P would terminate it

Ranges In Scansets

If you want to specify a range of characters, simply put a dash between the characters at either end of the range:

- e.g. “%[A-Z]” would accept any uppercase letters between A and Z inclusive
- e.g. “%[A-Za-z0-9]” would accept any uppercase or lowercase letters and any digits

To allow a dash character in the scanset, simply lead off with the dash:

- e.g. "%[-A-Za-z0-9]" would accept a dash, any uppercase or lowercase letters, and any digits

Scansets Are Limiting

The problem with scansets is that you have to strongly define the characters you are willing to accept.

Often, you know what you **don’t** want to accept.

- This is called “specifying the delimiter”.
- We can make use of this by using an “inverted scanset”.

Inverted Scansets

Inverted scansets allow you to specify the delimiter that ends a string:

- e.g. "%[^,]" will accept any character except a comma (or the end of the string)

These are used **much** more often than regular scansets, because you typically definitely know what you **don't** want to accept.

Using multiple consecutive
inverted scansets
for more complex data
can be complicated

Example is found at:

http://stackoverflow.com/questions/1508754/how-do-i-parse-out-the-fields-in-a-comma-separated-string-using-sscanf-while-sup

Make use of

%*c

which means

“ignore a single character”

```
char name1[kNameSize] = "";
char name2[kNameSize] = "";
char name3[kNameSize] = "";

printf("%d\n", scanf("%[^,]%*c%[^,]%*c%[^,]", name1, name2,
    name3));
printf("Name 1:%s\nName 2:%s\nName 3:%s\n", name1, name2,
    name3);
```

For more complex data,
there are other tools,
such as strtok()
(covered in the AST course)

Review!

What's a good way to get two words as input?

Review!

What's a good way to get a string that ends with a comma?

The Rest of scanf()

So far, we’ve only looked at format strings.

We also need to specify **where** we want the formatted input to go.

- In order to do this, we need to specify the addresses of the variables we want to fill in.

Similar to printf(), we do this after the format string:

- e.g. scanf("%d %d", &i, &j);

We **must** put the & in front of the variable names because we need the addresses.

We'll learn more about this when we take about pointers.

One Place Where We Don’t Need The & Operator

If you are getting a string, you can just give the name of the array:

- e.g. scanf(“%d %s %d”, &i, str1, &j);
- e.g. scanf(“%d %[^,] %d”, &i, str1, &j);

Failing On A Match

If scanf() fails on a match (e.g. you have a 'w' when it's expecting a digit), it stops scanning.

Any remaining variables left to fill in remain unchanged from what they were before the call.

scanf() returns the number of items it successfully matched (always between 0 and the maximum number of items in the format string).

It is common to have the scanf() call in an if statement to check to see if it succeeded.

Review!

If you’re trying to get a number from the user and they enter “55”, what value gets returned from scanf()?

OK, Now Why Is scanf() Not Used Much In “The Real World”?

If scanf() fails on a match, it stops getting input right at that point.

In that case, if you try to get input again, the scanning will start at the point it failed at, rather than at the next line that the user may have entered!

This will often create infinite loops.

So why do you find scanf() used in textbooks?

The writer of the textbook doesn’t have to maintain your code

The writer of the textbook wants something easy to use that they don’t have to explain much

The writer of the textbook doesn’t care about whether or not you write good code

The writer of the textbook doesn't care about whether or not you are employable

Example

```
while( i != 0 )
{
    printf("Enter two numbers\n");
    if( scanf("%d %d", &i, &j) < 2 )
            printf("Error: need two numbers\n");
    else
            printf("%d %d\n", i, j);
}
```

In this case, if we enter a non-digit, we will have an infinite loop because the non-digit will always remain as input, with scanf() failing every single time.

Review!

Why can there be an infinite loop in the previous example?

A Better Way

There is another function that is related to scanf(): sscanf().

sscanf() scans strings, instead of the keyboard.

It takes an additional string parameter **before** the format string.

e.g. sscanf(str1, “%d %d”, &i, &j);

Using fgets() and sscanf()

If we use fgets() to get a whole line of input from the user and sscanf() to scan that line once it's put in a string, we can be much safer.

Example

```
fgets(str1, kArraySizeStr1, stdin);
if( sscanf(str1, "%d", &i) != 1 )
{
    printf(“Need to enter a number\n”);
}
```

This Looks Like …

getNum()!

getNum()

```
int getNum(void)
{
char record[121] = {0};                                  /* record stores the string */
int number = 0;

    /* use fgets() to get a string from the keyboard */
    fgets(record, 121, stdin);
    /* extract the number from the string; sscanf() returns a number
     * corresponding with the number of items it found in the string */
    if( sscanf(record, "%d", &number) != 1 )
    {
            /* if the user did not enter a number recognizable by
             * the system, set number to -1 */
            number = -1;
    }
    return number;
}
```

So, There’s No Reason To Use scanf()!

Use fgets() with sscanf() instead of scanf().

If you choose to use scanf() anyway,

I will try to break it

If I succeed, you will lose

20 marks

DO YOU FEEL LUCKY.... PUNK?
WELL.... DO YA?

Other Input Functions That Don’t Work As Well As You Might Think

getc()

getchar()

And, finally, the "Press Any Key To Continue" function

getch() can be used

to get a key from the user

without requiring

the ENTER key

```
#include <conio.h> // needed
// stuff goes here
char ch = 0;
printf(“Press any key to continue”);
ch =  getch();
```

And, almost lastly, but VERY importantly!

Always prompt the user

Actually, **never** get user input without telling the user what you want first!

One last thing!

In the Computer Security course,

we'll learn about other options for safe input.

Use

#pragma warning(disable: 4996)

to avoid the Microsoft-specific warning about sscanf()

Your submitted programs must always compile without warnings

(we will recompile code occasionally)

Summary

1. User input is done with string functions.
2. scanf() is risky.
3. Use fgets() and sscanf() for safety.