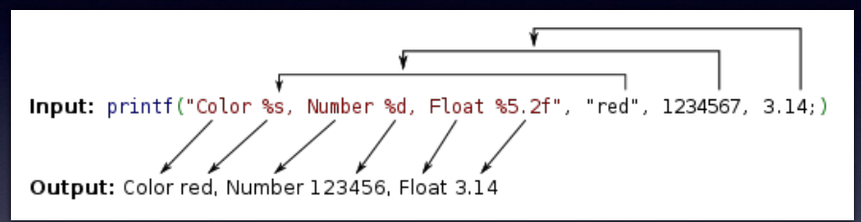
scanf()

Formatted output

- `printf()` can print out the value of variables (and text)
- Specify where the variables are placed in the output using conversion characters (e.g. `%d`)


Input: printf("Color %s, Number %d, Float %5.2f", "red", 1234567, 3.14;)
Output: Color red, Number 123456, Float 3.14

Formatted Input

- `scanf()` reads a series of values from the standard input (keyboard)
- Unlike `getchar()`, it attempts to understand the input
- Converts it to `chars`, `ints`, `floats`, `doubles` etc.
- Can read more than one value at a time

Input

- Could do this manually
- Read a character, convert to a number
- Multiply by ten and add the new character…
- But we tend to do it a lot in programs so C provides a library function to do it

Go build this

Simple `scanf`

- Need to tell `scanf` what we want it to read
- Uses the same syntax as `printf()`
- Pass a pointer to where we want it to store the value
- Usually the address of a variable
- Must be of the right type…

i.e.

```
int val;

scanf(“%d”, &val);
```

Read in an integer (because of %d -- store the result in val)

How it works

- `scanf()` takes two things
- A string telling it what type of things it is looking for
- A pointer to some memory where to store a value of that type
- `scanf()` then reads characters and converts them into whatever is requested

couldn’t use a global variable either -- how would scanf know which one you wanted to use???

How it works

- If it can't process the input, e.g. you type letters while trying to read an integer
- `scanf` can't return a value so the memory you point to will be left unchanged
- This begs the question:
  How can you tell if `scanf()` *read the same value or it didn't read anything at all?*

Go show what we mean with our scanf program

Count of values

- `scanf` also returns a value in the normal way (i.e. with a `return`)
- This is either, the number of values successfully read
- Or `-1` if the end of file was reached
- Check this to see if value correctly entered

```
int val;
int r;

r = scanf(“%d”, &val);
if(r == 0)
{
  printf(“Please enter an integer.\n”);
}
```

Read in an integer (because of %d -- store the result in val)

If

Interpolation String	`printf` meaning	`scanf` meaning	Type
`%d`	print in decimal	read in decimal	`int`
`%i`	print in decimal	read in decimal (can be in octal)	
`%o`	print in octal	read in octal	
`%x/%X`	printf in hex	read in hex	
`%u`	print unsigned	read unsigned	`unsigned int`
`%c`	print a single char	read a single char	`int/char`

%c takes an int for printf (C will automatically promote a char to an int when you call printf)

scanf takes %c to be a pointer to a char (i.e. char *)

Interpolation String	`printf` meaning	`scanf` meaning	Type
`%s`	print string	read string	`char *`
`%f`	print floating point number	read a `float`	`double/float`
`%lf`	print floating point number	read a floating point number	`double`
`%e,%E`	print in exponent form	read a float	`double/float`
`%g, %G`	print appropriately	read in a float	`double/float`
`%p`	print a pointer	N/A	`void *`
`%%`	print a '%'	a '%'	`N/A`

string is a series of character that ends with \0 (scanf expects the pointer to point to enough memory to read the string)

If two types, first is printf second is scanf

Give some demos...

Read multiple

- Just as `printf` can print multiple things out
- So `scanf()` can read in multiple values
- Give a format specification for each value in the string
- And pointers to store each value
- If you these don't match up, don't expect your program to work sanely…

these being the format specification and the string

Read Multiple Values

- Values need to be separable
- So two numbers would need to be separated by a space
- But a number followed by a character wouldn't need to be (unless the character was a digit)
- Treats any run of space characters as separator

space characters include tabs, and newlines

Go show some examples...

Formatted Input

- `scanf` is designed to support formatted input
- This means that we can describe how things like dates might be entered
- This is done by putting the characters in the specification
- Specified characters *must* be present

Reading a date

- `scanf("%d/%d/%d", &day, &month, &year);`
- Read an `int`
- Immediately followed by a `/`
- Immediately followed by an `int`
- Immediately followed by a `/`

Show how it works

and how if the input doesn't EXACTLY match that it doesn't produce the correct values

but the return value tells us how many it was able to read

Can build any input string we like

Miscellany

- `scanf` only consumes input that it can convert
- So if an invalid character is found it will be left to be read by something else
- What happens if you ask to read an `int` and give it a `float`?

`printf` specification

- `printf` lets you specify how things are printed in great detail by modifying the specification
- By inserting extra characters between the `%` and the character
- For example, `%5d` means print an integer in and make sure the value is *at least* 5 characters wide

`printf` specification

- Between the % and conversion character there may be in order:
  - A minus sign, specifies left adjustment
  - A number to specify minimum field width (padding if necessary)
  - A period to specify width from precision
  - A number which specifies the precision

`printf` specification

- Precision is the number of decimal places
- Or the maximum number of characters to print from a string
- Or the minimum number of digits for an integer