CS16, 10S, **Handout to go with H08** (command line arguments) (printable PDF)

Available online at: http://www.cs.ucsb.edu/~pconrad/cs16/10S/homework/H08/handout
The assignment is available at http://www.cs.ucsb.edu/~pconrad/cs16/10S/homework/H08

This handout is your reading assignment to go with H08—this material is not covered in the textbook.
This is a review of some material covered in lecture during the week of 04/05 through 04/09.
The material here is also covered in lab03.

Command line arguments:

Command line arguments allow us to provide input to a C program through the command line.

For example, instead of typing

```
./gameScore
```

to run the program foo.c, we type:

```
./gameScore Steelers 30 Dolphins 20
```

and the values `"Steelers"` `"30"` `"Dolphins"` and `"20"` will be available inside the C program—we don't have to use `scanf` to prompt for them.

Here's how it works:

- `argc` is the number of arguments on the command line. For example in the case of

  ```
  ./gameScore Steelers 30 Dolphins 20
  ```

  `argc` is equal to 5, because there are five things on the command line.

- `argv` is an array of `char *` values, `argv[0]`, `argv[1]`, `argv[2]`, etc. where each of those has the value of exactly one of the things on the command line.

  For example, in this case:

  `argv[0]` has the value `"./gameScore"`
  `argv[1]` has the value `"Steelers"`
  `argv[2]` has the value `"30"` (note that this is a string, a `char *`, not an `int`)
  `argv[3]` has the value `"Dolphins"`
  `argv[4]` has the value `"20"` (again, a string, a `char *`, not an `int`)

Please turn over for more

Continued from other side

Double Subscripting

Working again with the command line:
`./gameScore Steelers 30 Dolphins 20`

we see that `argv[1]` has the value `"Steelers"`, and `argv[3]` has the value `"Dolphins"`.

We can double subscript these, because `argv[1]`, as a string, is an array of characters followed by a null character.

That is, `argv[1][0]` is the character `'S'`, `argv[1][1]` is `'t'`, `argv[1][2]` is `'e'`, etc.

The full string is shown in this table:

argv[1][0]	argv[1][1]	argv[1][2]	argv[1][3]	argv[1][4]	argv[1][5]	argv[1][6]	argv[1][7]	argv[1][8]	argv[1][9]
'S'	't'	'e'	'e'	'l'	'e'	'r'	's'	'\0'	*invalid subscript*

Similarly for `argv[2]`, which is `"30"`, we have:

argv[2][0]	argv[2][1]	argv[2][2]	argv[2][3]
'3'	'0'	'\0'	*invalid subscript*

Converting to integer

To convert to integer, we use the function `atoi()` as shown below.

- We must use `#include <stdlib.h>` in our program before using atoi()
- We need to check the value of `argc` first
    - if we try to convert `argv[1]` using `atoi`, but when `argv[1]` doesn't have a value (because `argc<2`) then we'll get an error (often a "`segmentation fault`")

For example, this shows the proper way to check argc before we access `argv[1]` and `argv[2]`. Here, `argc` should be 3 (remember that the program name, `./makeBox` in this case, counts as one of the elements of `argv`)

```
int width, height;
if (argc!=3)
  {
     printf("Usage: ./makeBox width height\n");
     return 1;
  }
width=atoi(argv[1]);
height=atoi(argv[2]);
```

Converting to double (floating point numbers)

Converting to `double` works the same as integers, but we use `atof` instead of `atoi`.

- Note that `atof` returns a double, not a float—yet the name is `atof`, not atod. Go figure.
- You also need to `#include <stdlib>` to work with atof. If you don't, you'll get strange results (sometimes with no error message to warn you.)

End of H08 handout