1 Basic Syntax

Some of the data types:

int Declares an integer (it must be declared before it can be used).
float Declares a float: number with some decimal places.
double Declares a double: number with decimal places.
    Twice as many as in a float are possible.
char Declares a char (or character: a single letter or symbol).

Some useful commands:

printf Displays the input (with some changes).
#include Use the stuff found in the file indicated.
    You need #include <stdio.h> for printing things.
atoi Covert a char* to an int
atof Covert a char* to a float

Some basic math:

i++ Adds 1 to i (equivalent to i=i+1)
i+=7 Adds 7 to i (equivalent to i=i+7)
i/=9 Divides i by 9
(float)5/9 "Casts" 5/9 as a float. Since 5 and 9 are ints, this would normally come out as an int: 0.

1.1 printf

If you give a regular string to printf, you’ll get what you expect. If you type:

printf("Hi there!");
The result is:

"Hi there!".

However, you will often want to mix in the values of variables with our output. You can do this by putting % followed by a letter for the type of variable you want to put in. Then, after the string, you put in as extra parameters the variables to print out (in the same order).

Let’s say myLetter, myInteger, and myFloat are a char, int and float respectively. If you wanted to print them, you might type:

```c
printf("Letter: %c\n", myLetter);
printf("Integer: %d\n", myInteger);
printf("Float: %f\n", myFloat);
```

\n represents a newline character (so the output will be on several lines). The %c takes the place of our character, %d takes the place of our integer (we could have used %i instead), and %f holds the spot for a float, so our output would be something like:

Letter: f
Integer: 64
Float: 1.333

We could have done the same thing in one command with

```c
printf("Letter: %c\nInteger: %d\nFloat: %f", myLetter, myInteger, myFloat);
```

One last tricky thing about printf. Sometimes you will want to print a column of output. It will look funny if it is not aligned consistently. printf has a way around this. If the first thing after the % is a number instead of a letter, you can use it to insist that the variable going into that spot take up at least some number of spaces. For example

```c
printf("%6d",money);
```

This will print the int called money and put spaces in front of it to make it fill up as much space as six letters. If the variable is a float, you can also pick how many decimal places are shown:

```c
printf("%6.2f",money);
```

This will print the float called money, put spaces to make it take up six slots, and show 2 decimal places. This is demonstrated in the next example.
1.2 Example

The following program illustrates most of what we've learned so far. It also shows how to set up a main method. The main method is the one a compiler will turn into an executable if you sic it on the file.

```c
#include <stdio.h>

int main(){

    printf("Every age has a language of its own. This is it.\n");

    int i;

    for(i = 0; i < 4; i++){
        printf("The integer after %d is %d.\n", i-1, i);
    }

    int four = 4;
    if(four > 3){
        printf("%d is more than %d.\n", four, 3);
    }

    float money = 256;
    while(money != .25){
        printf("%6.2f\n", money);
        money /= 2;
    }
    return 0;
}
```

The main method is declared as an int because it will return an int when it is done. The method is also "seen" as an int by anything that invokes it. If the compiler says anything about an int called main, it’s because it sees main as an int.

Unlike some languages, the variable used in the for loop must be declared before it is used: you'll notice the statement int i; just before the loop in the example.

The statement return 0 ends the method. It also sends a 0 back to whomever called the method. When main returns 0, it indicates successful execution.

The output of this program is

Every age has a language of it’s own. This is it.
The integer after -1 is 0.
The integer after 0 is 1.
The integer after 1 is 2.
The integer after 2 is 3.
4 is more than 3.

256.00
128.00
64.00
32.00
16.00
8.00
4.00
2.00
1.00
0.50

2 Pointers

2.1 Syntax

char* Declares a pointer to a character (char).
This is also used to point to the beginning of arrays of chars.

int* Declares a pointer to an int.

p[i] Access the i\text{th} element in the array/pointer p
(or just the equivalent number of bytes after p).

p++ Increment p. p will point to the next item in the array if it is one.
Otherwise, it will just point to whatever happens to come next in memory.

*p Dereference p (The thing p points to).
If p is an int*, *p can be treated just like an int.

&i The address of i (or a pointer to i). So if i is an int:
int* p = &i;

‘\0’ The NULL character for terminating strings.
This is automatically added to a string that is defined by a constant:
p = “P is a char*; this string ends in a null character.”;

2.2 Explanation

Pointers are a powerful part of C. They are \textit{ints} that point to spots in memory where stuff is kept. A \textit{char*} is a pointer points to a \textit{char}. An \textit{Elephant*} points to an \textit{Elephant}. You can use a pointer to do things with the object it points to, but remember that it isn’t the object itself.

Pointers are also the way C makes arrays of data. More often than not, a \textit{char*} points to a place in memory where many \textit{chars} are lined up in a particular order. This is called a string. The last \textit{char} in a string should be ‘\0’; the null character. It marks the end of the string. Most programs assume that all strings end with ‘\0’. If it is not there, many programs will run off the end of the string and look at whatever happens to be stored next in memory. \texttt{printf}, for example, will print everything from the pointer’s target until the next null character it runs into. If \texttt{printf} prints lots of German characters, there’s a
good chance your string isn’t null-terminated.

2.3 Warning

Pointers are ints that point to things. chars are ints that can be treated as a character. Your program will gladly treat pointers and chars as ints if they are put in an int-like context. Similarly ints will be treated like chars or pointers in some contexts. Let the confusion begin.

There are lots of other mistakes to be made with pointers. One is forgetting that you are dealing with the memory at a very low level. If you tell a pointer to point at something new, you can lose track of the old thing. If you alter what p points at, anything else pointing at that will also be pointing at a changed thing.

2.4 A simple program making use of pointers

```c
int main(int cargs, char** vargs){

    printf("I got passed %d parameters.\n", cargs);

    int i;
    for(i = 0; i < cargs; i++){
        printf("Parameter %d is %s\n", i, vargs[i]);
    }

    char* pointer = malloc(sizeof(char)*21);
    strcpy(pointer,"He's pointing at me!");

    printf("%s\n",pointer);

    char* middle = pointer + 5;

    printf("%s\n",middle);
    printf("%c\n",pointer[5]);

    *middle = 'j';

    printf("%s\n",middle);
    printf("%s\n",pointer);

    return 0;
}
```

You’ll notice this example has extra stuff in the parenthesis after main. Normally, main methods take two parameters: an int and a char pointer pointer
(char**). The char** points to the beginning of an array of char pointers. Each of these char* points to an array of chars. The int that got passed first is the number of char pointers in the array. Simplest thing in the world.

Let’s say we ran this program from the command line by typing

```
programName argument1 argument2 sloppy joe
```

The int received by main would actually be 5 because the name of the program is included as a parameter.

The output of the program would be

```
I got passed 5 parameters.
Parameter 0 is programName
Parameter 1 is argument1
Parameter 2 is argument2
Parameter 3 is sloppy
Parameter 4 is joe
He’s pointing at me!
pointing at me!
p
jointing at me!
He’s jointing at me!
```

3 Memory Management

If you are going to have very large structures, you will need to take advantage of heap memory. Normally, variables get put in the small, fast part of memory called the stack. If you try putting too much on the stack, your program will crash, no survivors.

The heap has as much space as you’re likely to need. To tell C to put something there, you use the command malloc. For example

```
int* p = malloc(8);
```

This makes p an int* that points to 8 free bytes of heap space. If you don’t know the size of the thing you want to store, use sizeof:

```
int* p = malloc(sizeof(int));
```

This gives you enough space to store an int. If you need to store an array of 12 ints:

```
int* p = malloc(sizeof(int)*12);
```

Using malloc, you can access a huge amount of memory. However, things on the heap don’t go away on their own. You must delete them yourself. You
probably won’t run out of room even if you don’t, but you might, and if you’re going to be good at this, you need to know how.

Freeing memory is pretty easy: `free(p)` frees up the memory p points to.

Freeing memory also means you no longer own that memory. If you try putting things there again, they may get stomped on by the operating system, and you will be wildly confused when your data keeps disappearing. For this reason, it is wise to assign freed pointers to something else. If another object is replacing the one that got freed, assign your pointer to point at the new object. If you’re getting rid of the object without replacing it, there is nowhere safe to point it. You can render it harmless by setting it to NULL:

```c
free(p);
p = NULL;
```

//NULL traditionally refers to pointers.
//However, next two calls mean the same thing:

```c
p = 0;
p = '\0';
```

After you make a pointer NULL, it can’t access anything. If you try, your program will crash—which is better than letting it carry on thinking it was okay to use that pointer.

### 4 Divide and Conquer

One key to good programming is splitting up identifiable tasks into separate methods. These should in turn be split up into separate files and directories as appropriate.

For example

```c
#include <stdio.h>

float getMean(int nums[], int size);

int main(int cargs, char** vargs){
    int i;

    int nums[cargs];

    for(i = 1; i < cargs; i++){
        int n = atoi(vargs[i]);
        nums[i-1] = n;
    }
    float mean = getMean(nums, cargs-1);
```
Calculating the mean of an array of numbers is an identifiable task. It can easily be turned into its own method.

You'll notice that `getMean` is declared before it is used by `main` but defined after. It could also just get defined before (which counts as a declaration). If you like to have your main method before the others, you'll need to put declarations first.

If you have a large number of helper methods, this can get messy. To avoid this, you can put all those declarations in a separate file (called a header file) and import it. For our example, we could put `float getMean(int nums[], int n)` in a separate file called `mean.h` and add the line `#import "mean.h"` to the `.c` file (use `""` instead of `<>` since it's your own file rather than a standard one).

You can get some strange results after including your own files if you do it wrong. When files are included, their contents take the place of the `#include` line. The compiler occasionally accuses a file of having an error when the problem lies in the way it includes other files.

5 Homework

Write a program to convert Fahrenheit temperatures to Celsius (or vice versa). Your program should accept temperature(s) to convert from the command line. Try formatting the output a few different ways. Try to use `printf` to arrange it nicely:

<table>
<thead>
<tr>
<th>F</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>32.0</td>
<td>0.0</td>
</tr>
<tr>
<td>212.0</td>
<td>100.0</td>
</tr>
</tbody>
</table>
5.1 Hints

You will want to convert inputs to floats, doubles or ints with atof or atoi. If you use atof, you will need to #include <stdlib.h>.

If F is the temperature in Fahrenheit and C is the temperature in Celsius, then

\[ F = \frac{9}{5}C + 32 \]
\[ C = \frac{5}{9}(F - 32) \]

www.cplusplus.com/reference may be useful. You can read about many of the standard library functions there. You shouldn’t need them for the assignment, but you should eventually learn the basic string manipulation methods like strcmp, strstr and strcpy.

If you get stuck trying to write this, ask somebody smart but not Jonathan Baker at motmahp@gmail.com. I know nutzing.