L2&3 - Fundamental Concepts

November 8, 2019

1 Introduction to C: Fundamental Concepts

1.0.1 PASS

Sean Kirkby - SA402 Tuesday 6:00 to 7:30pm and PA218 Thursday 6:00 - 7:30, office hours Monday 3:30-4:30 in the 4th floor of the library

1.0.2 Midterm

Saturday November 2 2019, 1:00-2:30 (tentative)

1.0.3 Lab Schedule

Attendance at lab session is not mandatory but may be the difference between *marginal* and *unsat*. Do not attend lab sections that are not your own.

1.1 Hello World in C

```
In [1]: /* Hello World in C */

        #include <stdio.h>
        #include <stdlib.h>

        int main(void){
            printf("Hello World\n");
            return EXIT_SUCCESS; /* */
        }
```

```
Hello World
```

Let's unpack that a bit **stdio.h** - contains I/O functions like *printf*
EXIT_SUCCESS - a reserved word, defined in **stdlib.h**. To indicate failure, return **EXIT_FAILURE**.

Every standard C program has exactly one function named *main*, which is the program's entry point. Passing *void* means that the function has no *parameters*; that is, the function takes no other information to run.

printf - send *formatted* output to the standard output stream (the console)

The returned value from *main* is the program's exit status; in this case, a macro from stdlib.

1.2 Fahrenheit to Celsius

In [1]:
```
#include <stdio.h>
#include <stdlib.h>

int main(void){
    int lower, upper, step;
    float fahr, cels;

    // Set up the iteration limits and step size
    lower = -100;
    upper = 220;
    step = 20;
    fahr = lower;

    while (fahr <= upper){
        cels = (fahr - 32) * 5. / 9.;
        printf("%4.0f %6.1f \n", fahr, cels); // note float formatting
        fahr = fahr + step;
    }
}
```

```
-100  -73.3
 -80  -62.2
 -60  -51.1
 -40  -40.0
 -20  -28.9
   0  -17.8
  20   -6.7
  40    4.4
  60   15.6
  80   26.7
 100   37.8
 120   48.9
 140   60.0
 160   71.1
 180   82.2
 200   93.3
 220  104.4
```

Let's unpack *that* a bit **=** - the *assignment* operator. The expression on the right hand side is stored into the variable on the left hand side.
- - subtraction
/ - division
** * ** - multiplication
Parentheses operate algebraically; code inside the parentheses executes first.

We are not concerned about numerical precision in this code as we will only print values to one decimal place.

Importance of style By some reports, 70% of program development cost can be spent on maintenance, so it's important to keep our codebase readable.
It is not required by the compiler to indent code that is within the braces, but it is good style.
* Put spaces around operators to make code easier to read. * All statements do requrire a semicolon to terminate. * Use meaningful and descriptive variable names * Put in lots of comments to describe what we are doing

Some *printf* conversion specifications **%f**: floating point number to full stored precision
%3.2f: floating point number with 3 digits before and 2 digits after the decimal point
%4.0f: floating point number with 4 digits before and no digits after the decimal point
%d: integer number

Datatypes

float A floating point type. Single-precision.

double Double-precision integer. Stores a variable to higher precision than the *float* type.

int Signed integer type; no decimal places. With signing, can represent a positive or negative number.

const Tells the compiler that this value will not change during program execution.
Dividing or dividing an *int* and *float* will always generate a float, rather than truncating.

While loop The while loop will iterate so long as the condition inside the brackets is *true*. Note that we must change something in the loop so as to make iteration non-infinite.

1.2.1 An experiment in truncation

In [5]:
```
#include <stdio.h>

int main(void){

    // make a floating point variable
    float rootbeer = 3.14159;

    // storing in an int truncates
    int eger = rootbeer;
    printf("%i", eger);
}
```

```
3
```

In [3]:
```
# include <stdio.h>

int main(void){
    float one, two, three, four, five, six;
    int fahr = 77;

    // these are all correct
    one = 5.0 / 9.0 * (fahr - 32);
    two = 5 / 9.0 * (fahr - 32); // int / float = float; float * int = float
    three = 5.0 / 9 * (fahr - 32); // float / int = float; float * int = float
    four = (fahr - 32) * 5 / 9; // float * int = float; float / int = float

    // these are  all wrong
    five = 5 / 9 * (fahr - 32); // int 5/9 is zero!!!
    six = (fahr - 32) * ( 5 / 9 );

    // outputs
    printf("%4.1f\n", one);
    printf("%4.1f\n", two);
    printf("%4.1f\n", three);
    printf("%4.1f\n", four);
    printf("%4.1f\n", five);
    printf("%4.1f\n", six);
}
```

```
25.0
25.0
25.0
25.0
 0.0
 0.0
```

1.2.2 Functions

In [13]:
```
/* Raise base to the n'th power, n >= 0. */

 # include <stdio.h>
 # include <stdlib.h>

 int power(int base, int n){

     int i, pow;

     pow = base;

     for ( i = 1; i <= n; n = n + 1){
         pow = pow * base;
```

```
    }

    return pow;
}

int main (void){
    int y;

    y = power(3, 3);

    printf("%d", y);
}
```

```
954437177
```

1.2.3 Function Prototypes

A function in C must be declared before it can be used. However, we can use a *function prototype* instead of the entire function before the first function call.

A variable need not have the same name in the function prototype as it has in the function declaration. In C, function arguements are passed by value - the parameters are local variables that are initialized with the values of the arguments.

In [13]:
```
/* Raise base to the n'th power, n >= 0. */
# include <stdio.h>
# include <stdlib.h>

int power (int, int); // function prototype

int main(void){
    int i;
    i = 0;

    while (i < 10){
        printf("%d %d %d\n", i, power(2,i), power(-3,i));
        i += 1;
    }

    return EXIT_SUCCESS;
}

int power(int base, int n){

    int i = n;
    int pow = 1;

    while ( i > 0){
        pow = pow * base;
```

```
        i -= 1;
    }

    return pow;
}
```

```
0 1 1
1 2 -3
2 4 9
3 8 -27
4 16 81
5 32 -243
6 64 729
7 128 -2187
8 256 6561
9 512 -19683
```

In C, function arguments are passed by value in general (except for arrays). Assigning a value to a parameter does not modify the corresponding argument. This can lead to tighter code. Incrementing or decrementing a local variable will not modify the variable passed into the function.

In [14]:
```
// we don't really need i; let's get rid of it.
#include <stdlib.h>
#include <stdio.h>

int power(int m, int n);

int main(void){
    int i = 0;
    while (i < 10){
        printf("%d %d %d\n", i, power(2,i), power(-3,i));
        i += 1;
    }
    return EXIT_SUCCESS;
}

int power(int base, int n){
    int pow = 1;
    while (n > 0){
        pow *= base;
        n -= 1;
    }
    return pow;
}
```

```
0 1 1
1 2 -3
2 4 9
```

```
3 8 -27
4 16 81
5 32 -243
6 64 729
7 128 -2187
8 256 6561
9 512 -19683
```

In [20]:
```c
// express as for loops instead
#include <stdlib.h>
#include <stdio.h>

int power(int m, int n);

int main(void){
    int i = 0;
    for (i = 0; i < 10; i++){
        printf("%d %d %d\n", i, power(2,i), power(-3,i));
    }
    return EXIT_SUCCESS;
}

int power(int base, int n){
    int pow = 1;
    for (; n > 0; n=n - 1){ // we don't need an initialization here; so we don't
        pow *= base;
    }
    return pow;
}
```

```
0 1 1
1 2 -3
2 4 9
3 8 -27
4 16 81
5 32 -243
6 64 729
7 128 -2187
8 256 6561
9 512 -19683
```

In [29]:
```c
// this is a bad program and you should feel bad
# include <stdlib.h>
# include <stdio.h>

int main(void){
```

```
            int i = 0;
            for (;;){ // yup, this garbage compiles
                printf("%d ", i);
                if ( i > 55 ){
                    return EXIT_SUCCESS;
                }
                i++;
            }
        }
```

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34
```

```
In [1]: // express as for loops instead
        #include <stdlib.h>
        #include <stdio.h>

        int power(int m, int n);

        int main(void){
            int i = 0;
            for (i = 0; i < 10; i++){
                printf("%d %d %d\n", i, power(2,i), power(-3,i*-1));
            }
            return EXIT_SUCCESS;
        }

        int power(int base, int n){

            if (n < 0){
                abort();
            }

            int pow = 1;
            for (; n > 0; n=n - 1){ // we don't need an initialization here; so we don't
                pow *= base;
            }
            return pow;
        }
```

```
0 1 1


[C kernel] Executable exited with code -6
```