Numeric Data Types in C

...and a couple of unusual operators

reference

- Kernighan & Ritchie -
  - Chapter 2, Types, Operators, and Expressions
(Binary) Integers - Counting

- `int`, `unsigned`
  - a.k.a. `signed int`, `unsigned int`

- `long`, `long unsigned`
  - a.k.a. `long int`, `long unsigned int`

- `short`, `short unsigned`

- `char`, `unsigned char`
  - a.k.a. `signed char`, `unsigned char`
  - Yes, you can do arithmetic with char's!

- `long long int`, `long long unsigned int`
  - Not on all systems

Integer Arithmetic

- The usual operators
  - `+`, `-`, `*`, `/`, `%`
  - Beware of integer division!
  - No "power" operator – that is a "scientific" math function

- Bit-oriented operators
  - Bit-shifting: `>>`, `<<`
  - Bit-wise Boolean operations: `&`, `|`, `^`, `~`
**Character arithmetic**

- This routine gets a character, does some arithmetic on it, and displays the result as both a character and a number.

- The numeric value turns out to be the ASCII value of the character.

```c
/* chars as ints */
#include <stdio.h>

int main(int argc, char **argv)
{
    char c;
    char d;

    printf("? ");
    c = getchar();

    d = c + 5;

    printf("%c %d %x\n", c, c, c);
    printf("%c %d %x\n", d, d, d);

    return 0;
}
```

**How big are numbers?**

- Different machine wordsizes result in different ranges for each data type.

- The limits.h header file defines the minimum and maximum value for each data type, on any particular machine.
  - `CHAR_MIN, CHAR_MAX, UCHAR_MAX`
  - `SHRT_MIN, SHRT_MAX, USHRT_MAX`
  - `INT_MIN, INT_MAX, UINT_MAX`
  - `LONG_MIN, LONG_MAX, ULONG_MAX`
  - `LLONG_MIN, LLONG_MAX, ULLONG_MAX`

- Why no UCHAR_MIN, … ULLONG_MIN ?
Reals - Scientific

- **float, double**
- **long double**
- Real operators: + - * /
- Scientific functions
  - `#include <math.h>`
  - compile with `-lm` flag
    - gcc -Wall -o foo foo.c -lm
    - the flag must go last on the line
  - Many functions available

Project – sine-cosine plotter

- Function to draw line proportional to numeric value
- “Beat” value - multiplication factor for cosine() function
  - y = sin(x) * cos(beats*x)
Sine-cosine plotter

- Features:
  - Prototype
  - Cmd-line arguments
  - Math library
  - Non-zero return value

(2013 version)

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

void drawline(double f, int max);

int main(int argc, char **argv)
{
  unsigned npoints, i;
  unsigned linelength;
  double bext;
  double x, y;
  int point;  
  if (argc < 3) {
    printf("usage: %s <npoints> <bext> <linelength>\n", argv[0]);
    return 1;
  }
  npoints = strtol(argv[1], NULL, 10);
  bext = strtol(argv[3], NULL, 10);
  linelength = strtol(argv[5], NULL, 10);
  for (i = 0; i < npoints; i++) {
    x = i / 5.0;
    y = sin(x) * cos(bext * x);
    drawline(y, linelength);
  }
  printf("\n");
  drawline(1, linelength);
  return 0;
}
```

This function abstracts out the work of displaying a value as a line of dashes, from the work of calculating the value.

- Features:
  - Initial value for a local variable
  - Conversion specifier with width.precision modifier
**Type Conversions**

- C uses "casts" to convert integer types to real types and vice versa

- Examples:
  - `char c = 'a'; double realChar; realChar = (double)c;`
  - `unsigned intPi = (unsigned)3.14159;`
  - `long int x;
    unsigned y;
    x = (long int)y;`

  Some casts are necessary; some casts aren't really needed, because the compiler can "do the right thing" without any help.

**Boolean**

- There is no Boolean type
- Zero (0) means "false"
- Non-zero means "true"
  - Logical operators produce One (1) for "true"
- Boolean values are used in if-else statements, loops, and the ternary operator
- Boolean values may be used in binary arithmetic operations (if you want to write obfuscated code)
Boolean Operators

- Logical
  - And: `&&`
  - Or: `||`
  - Not: `!`

- Relational
  - less than: `<`, less than or equal to: `<=`
  - greater than: `>`, greater than or equal to: `>=`
  - equal to: `==`, not equal to: `!=`

Assignment Operators

- Assignment in C is an operation that returns a value (like any other operation), and has the side-effect of changing the value of its left-hand operand.

- Available operators
  - assign:
    - `=`
  - arithmetic-assign:
    - `+= -, -= *= /= %=`
    - `&=` `^=` `|= <= >=`
The Ternary Operator

- Like an if-else block that returns a value
- Operator: `? :`
- Example:
  - `a = ( b > c ? 2*b : c/2 )`
    - gives `a` the value `2*b` or `c/2`, depending on whether `b` is greater than `c` or not

Project: hailstone sequence

The Hailstone sequence

- Collatz’s conjecture:
  - For starting value `m`, this sequence always reaches 1 in a finite number of steps.
- True?

```c
/* hailstone sequence */
#include <stdio.h>
#include <stdlib.h>

unsigned hailstone(unsigned n)
{
    /*
    if (n%2 == 0)
        return n/2;
    else
        return 3 * n + 1;
    */
    return (n%2 == 0 ? n/2 : 3*n + 1);
}

int main(int argc, char **argv)
{
    unsigned m;
    m = strtoul(argv[1], NULL, 10);
    printf("%u\n", m);
    while (m != 1) {
        m = hailstone(m);
        printf("%u\n", m);
    }
    return 0;
}
```
**The Comma Operator**

- The comma operator, or sequencing operator, performs operations in a left-to-right manner, and returns the value of the right-most operation.

- This also makes sense if the operations have side effects (e.g. assignments). Otherwise, you might as well use separate statements.

- Example: swap a and b, using c
  
  ```
  c = a , a = b , b = c ;
  ```

---

**Example: Bubble Sort**

- In a Bubble sort, adjacent elements are compared, and swapped if necessary.

- The comparisons start at one end, step to the other end; at which point the largest (or smallest) value has “bubbled” to its position.

- This process is repeated until no further swaps occur, at which point the elements are sorted.

- It is generally considered an inefficient sort; but it is easy to code. And it illustrates some programming features.

- So let’s try it.
**Bubble Sort - details**

- Generate some random numbers between 0 and 1
  - Store in an array
  - Use a `#define` to parameterize the array size
- After each bubblepass, display the array using the `linedraw()` function
  - Makes visualization easier
  - Also use a `#define` for the desired row length
- Use a do{}while loop to perform bubblepasses
  - At least one pass
  - Test on occurrence of swap(s), reported by `bubblepass()`

### bubblesort solution

```c
/* BUBBLESORT example
   also demonstrates use of random numbers (again),
   and use of preprocessor defines.
   2013-09-10 */
#include <stdio.h>
#include <stdlib.h> // random, etc.
#include <time.h>

#define ARRAYSIZE 20
#define LIMELNGTH 40

void fill_array(int n, unsigned size);
void drawdata(int n, unsigned size);
void linedraw(double f, int max, char mark);

int main(int argc, char **argv)
{  
    int numbers[ARRAYSIZE];
    fill_array(numbers, ARRAYSIZE);
    bubblesort(numbers, ARRAYSIZE);
    return 0;
}

void fill_array(int n, unsigned size)
{ 
    unsigned i;
    randize(size); // for(i = 0; i < size; i++)
        n[i] = rand() - RAND_MAX/2;
}

void bubble_sort(int n, unsigned size)
{ 
    unsigned i, j, tmp, sorted;
    for (sorted = 0; i < size; i++)
    { 
        for (j = 0; j < size - i;)
        { 
            if (n[j] > n[j + 1])
            { 
                tmp = n[j];
                n[j] = n[j+1];
                n[j+1] = tmp;
                sorted = 0;
            }
            j++;
        }
        if (sorted)
        { 
            printf("%s Pass %d,%d\n", "$", i, sorted);
            drawdata(n, ARRAYSIZE);
        }
    }
}

void drawdata(int n, unsigned size)
{  
    unsigned j, junk;
    for (j = 0; j < size; j++)
    { 
        linedraw(2.0*gety(j)/LIMELNGTH, ' '); 
        junk = getchar();
    }
}

void linedraw(double f, int max, char mark)
{ 
    double scaled = (f + 1)/2 * max;
    int c;
    for(c = 0; c < scaled; c++)
    { 
        putchar(mark);
        printf("%d", c);
        putchar(mark);
    }
}
```

```c
//------------------------------
void drawdata(int n, unsigned size)
{ 
    unsigned j, junk;
    for (j = 0; j < size; j++)
    { 
        linedraw(2.0*gety(j)/LIMELNGTH, ' '); 
        junk = getchar();
    }
}
```

```c
//------------------------------
void linedraw(double f, int max, char mark)
{ 
    double scaled = (f + 1)/2 * max;
    int c;
    for(c = 0; c < scaled; c++)
    { 
        putchar(mark);
        printf("%d", c);
        putchar(mark);
    }
}
```