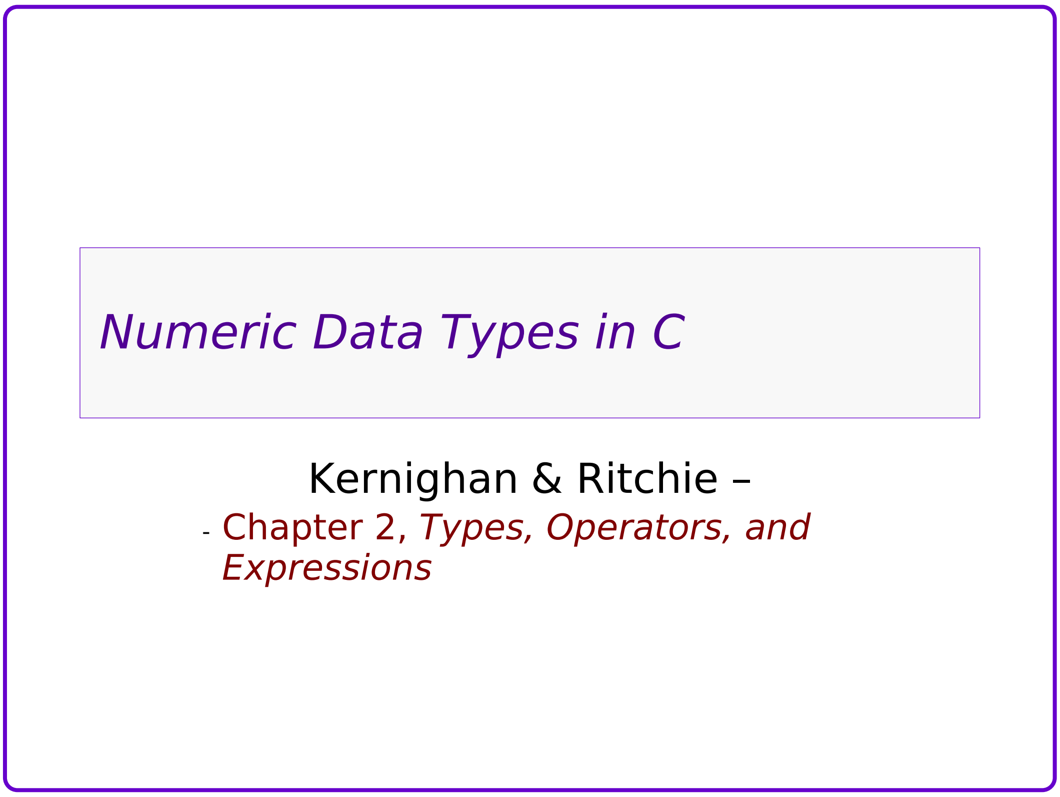
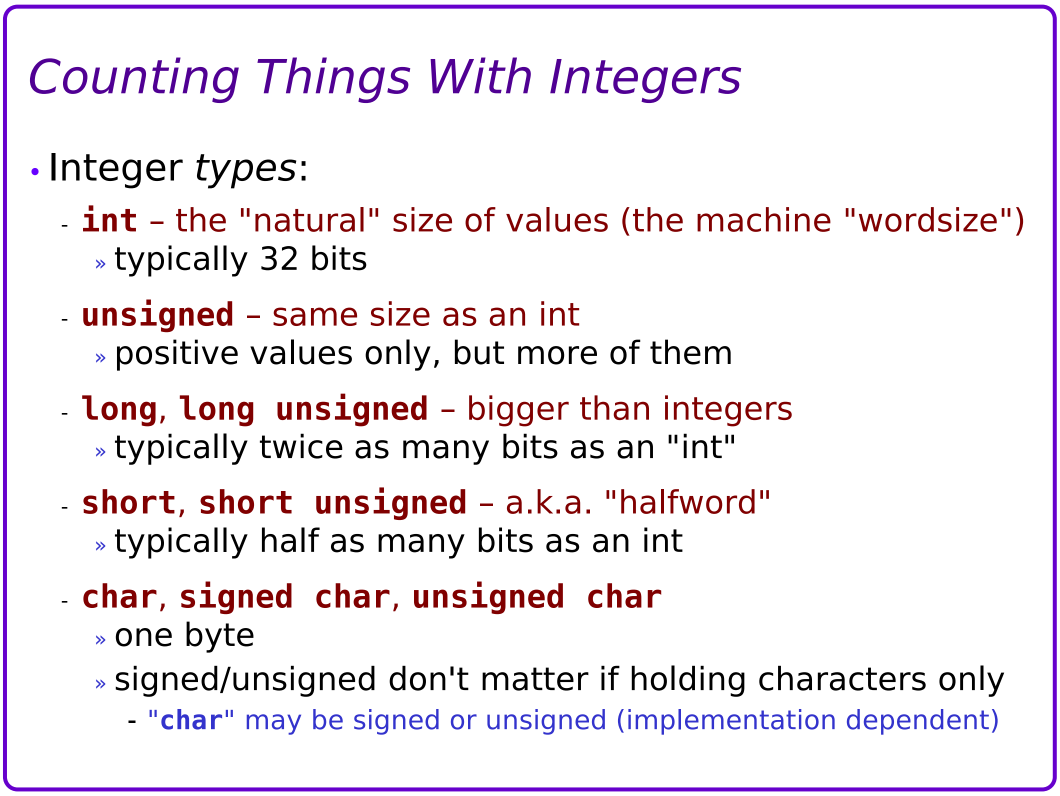


Numeric Data Types in C

Kernighan & Ritchie –

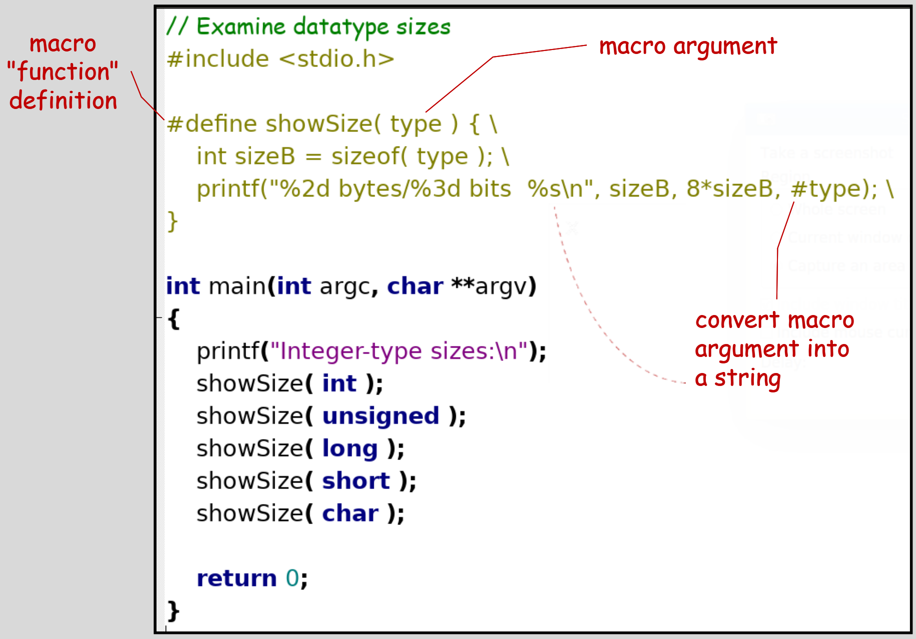
- Chapter 2, *Types, Operators, and Expressions*



Counting Things With Integers

- Integer *types*:
  - **`int`** – the "natural" size of values (the machine "wordsize")
    - » typically 32 bits
  - **`unsigned`** – same size as an int
    - » positive values only, but more of them
  - **`long`**, **`long unsigned`** – bigger than integers
    - » typically twice as many bits as an "int"
  - **`short`**, **`short unsigned`** – a.k.a. "halfword"
    - » typically half as many bits as an int
  - **`char`**, **`signed char`**, **`unsigned char`**
    - » one byte
    - » signed/unsigned don't matter if holding characters only
      - "**`char`**" may be signed or unsigned (implementation dependent)

Code Activity



```
// Examine datatype sizes
#include <stdio.h>

#define showSize( type ) { \
    int sizeB = sizeof( type ); \
    printf("%2d bytes/%3d bits  %s\n", sizeB, 8*sizeB, #type); \
}

int main(int argc, char **argv)
{
    printf("Integer-type sizes:\n");
    showSize( int );
    showSize( unsigned );
    showSize( long );
    showSize( short );
    showSize( char );

    return 0;
}
```

Code Discussion - Macros

- "#" introduces a *preprocessor macro*
  - "#" appears at beginning of the line
  - Macro continues to the end of the line
  - Multiline macros must use line-continuation character "\" at end of each line!
- Not part of the C language itself
  - Text expansion *before* compiling step
- "#include" – substitutes an entire header file into the source file
  - Just as if you did a manual "copy-paste"

Macros with Arguments

- Macro "function" substitutes body into the file
  - Arguments replaced into the text expansion
  - Comparable to C++ "inline function"
- *String* expansion –converts macro argument into a text string
  - "#" *stringize* operator applied to argument
  - *Stringize* operator is distinct from the leading "#"
- Example: the showSize() macro function
  - Argument "type" used as a datatype in "`sizeof()`"
    - » ...`sizeof()` is a macro itself, but part of C language
  - "#type" converts type into a labelling string

Why Use a Macro Instead of a Function?

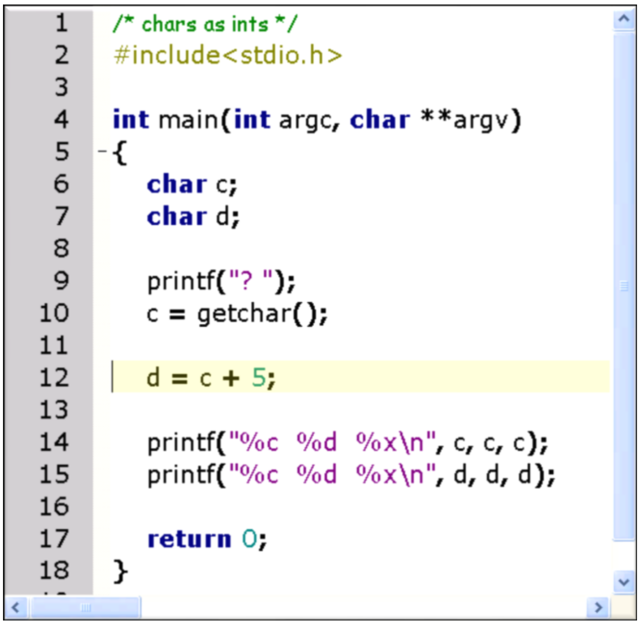
- In this case, the argument is a *data type* instead of a variable or constant value
- Functions cannot accept data types as arguments
  - "`sizeof()`" is a builtin *operator* in C
- *Preprocessor* macros are expanded *before* compilation
  - Expansion treats macro body, arguments as merely pieces of text

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: **& | ^ ~**
  - (These aren't strictly arithmetic operators, but they have an effect on the arithmetic values of variables)

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.



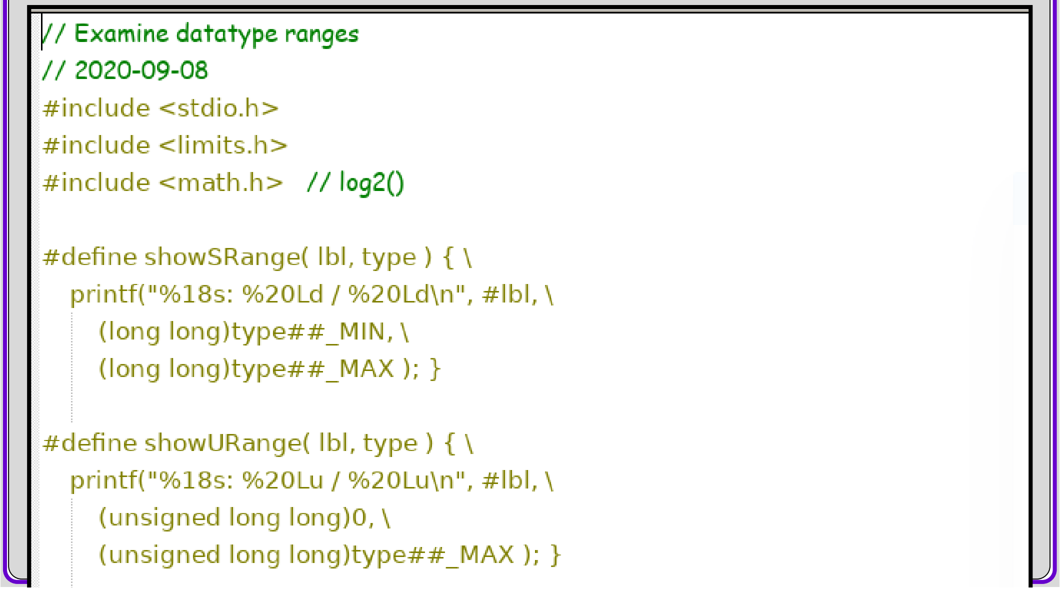
```c
/* chars as ints */
#include<stdio.h>

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 the 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, UINT_MIN, etc. ?

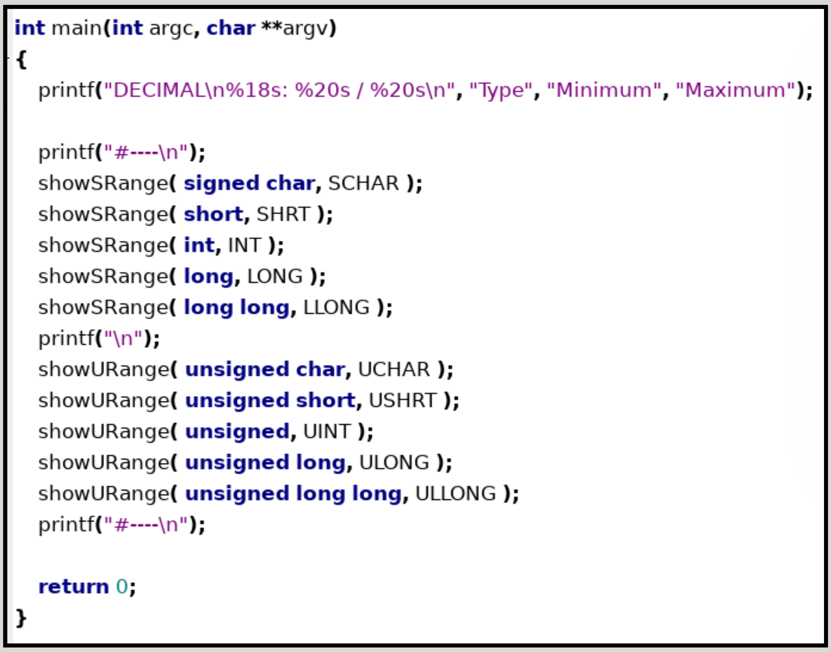
Look at datatypes again (A):

- Two more macros – print minima, maxima



```
// Examine datatype ranges
// 2020-09-08
#include <stdio.h>
#include <limits.h>
#include <math.h>   // log2()

#define showSRange( lbl, type ) { \
   printf("%18s: %20Ld / %20Ld\n", #lbl, \
      (long long)type##_MIN, \
      (long long)type##_MAX ); }

#define showURange( lbl, type ) { \
   printf("%18s: %20Lu / %20Lu\n", #lbl, \
      (unsigned long long)0, \
      (unsigned long long)type##_MAX ); }
```

Look at datatypes again (B):



```
int main(int argc, char **argv)
{
   printf("DECIMAL\n%18s: %20s / %20s\n", "Type", "Minimum", "Maximum");

   printf("#----\n");
   showSRange( signed char, SCHAR );
   showSRange( short, SHRT );
   showSRange( int, INT );
   showSRange( long, LONG );
   showSRange( long long, LLONG );
   printf("\n");
   showURange( unsigned char, UCHAR );
   showURange( unsigned short, USHRT );
   showURange( unsigned, UINT );
   showURange( unsigned long, ULONG );
   showURange( unsigned long long, ULLONG );
   printf("#----\n");

   return 0;
}
```

Reals - Scientific

- **float**, **double**
- **long double**
- Real operators: **+ - * /**
- Scientific functions
  - pow(), sin(), exp(), log(), ...
    many functions available

  - **#include <math.h>**
  - compile with "**-lm**" flag
    - gcc -Wall -o foo foo.c   **-lm**
    - the flag must go last on the line

Reals - Scientific

- **float**, **double**
  -10.05
- **long double**
  3.1415926535…
- Real operators:
  + - * /

long double pi = 3.1415926535…

long double area = pi * r*r

- Scientific functions
  - **#include <math.h>**
  - pow(), sin(), exp(), log(), … *many* functions available
  - Constants: **π** (as M_PI), **e** (as M_E), **√2** (as M_SQRT2), etc.
  - compile with "**-lm**" flag
    - gcc -Wall -o foo foo.c **-lm**

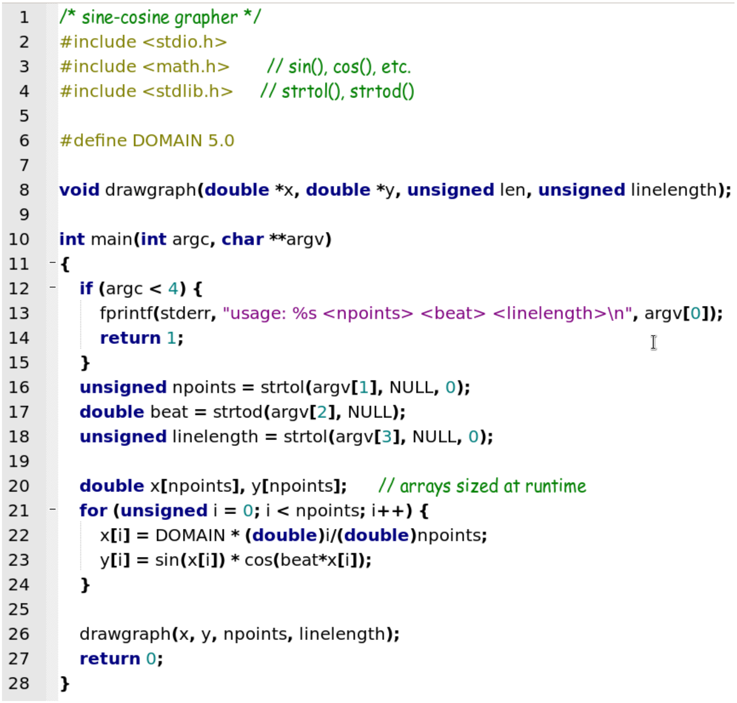
Project – sine-cosine plotter

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

Sine-cosine plotter

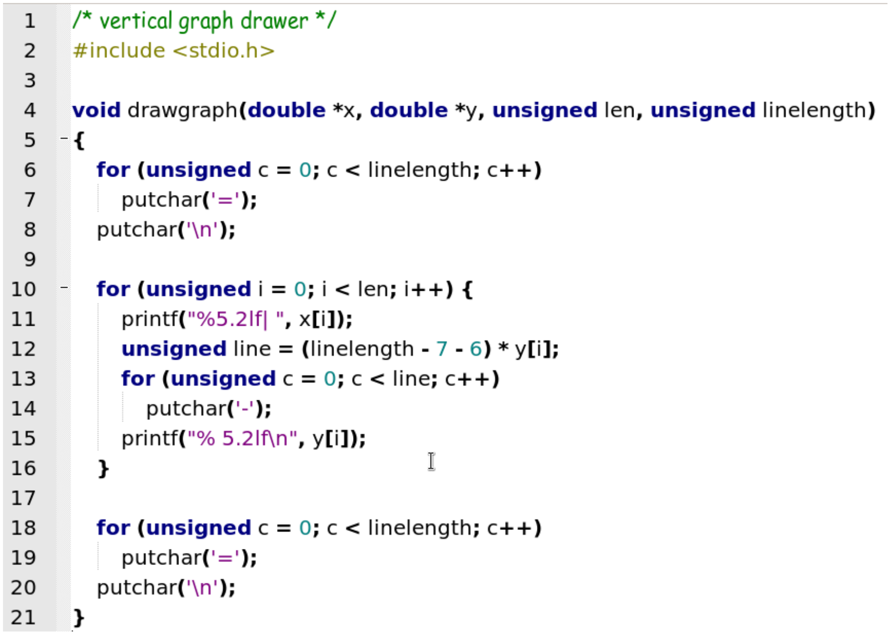
- Features:
  - Prototype
  - Cmd-line arguments
  - Math library
  - Non-zero return value

(2019 version)



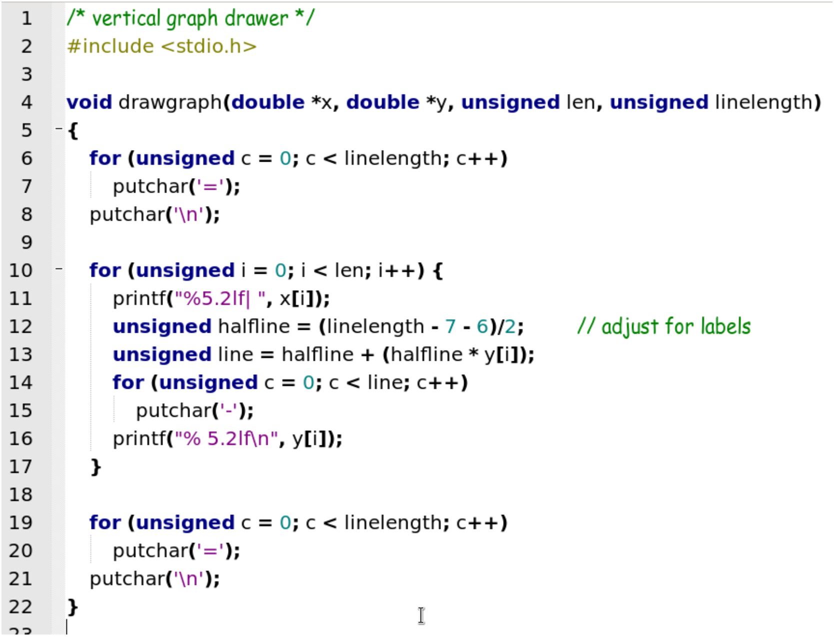
```
/* sine-cosine grapher */
#include <stdio.h>
#include <math.h>       // sin(), cos(), etc.
#include <stdlib.h>     // strtol(), strtod()

#define DOMAIN 5.0

void drawgraph(double *x, double *y, unsigned len, unsigned linelength);

int main(int argc, char **argv)
{
    if (argc < 4) {
        fprintf(stderr, "usage: %s <npoints> <beat> <linelength>\n", argv[0]);
        return 1;
    }
    unsigned npoints = strtol(argv[1], NULL, 0);
    double beat = strtod(argv[2], NULL);
    unsigned linelength = strtol(argv[3], NULL, 0);

    double x[npoints], y[npoints];      // arrays sized at runtime
    for (unsigned i = 0; i < npoints; i++) {
        x[i] = DOMAIN * (double)i/(double)npoints;
        y[i] = sin(x[i]) * cos(beat*x[i]);
    }

    drawgraph(x, y, npoints, linelength);
    return 0;
}
```

drawgraph, first try



```
/* vertical graph drawer */
#include <stdio.h>

void drawgraph(double *x, double *y, unsigned len, unsigned linelength)
{
    for (unsigned c = 0; c < linelength; c++)
        putchar('=');
    putchar('\n');

    for (unsigned i = 0; i < len; i++) {
        printf("%5.2lf| ", x[i]);
        unsigned line = (linelength - 7 - 6) * y[i];
        for (unsigned c = 0; c < line; c++)
            putchar('-');
        printf("% 5.2lf\n", y[i]);
    }

    for (unsigned c = 0; c < linelength; c++)
        putchar('=');
    putchar('\n');
}
```

drawgraph, second try

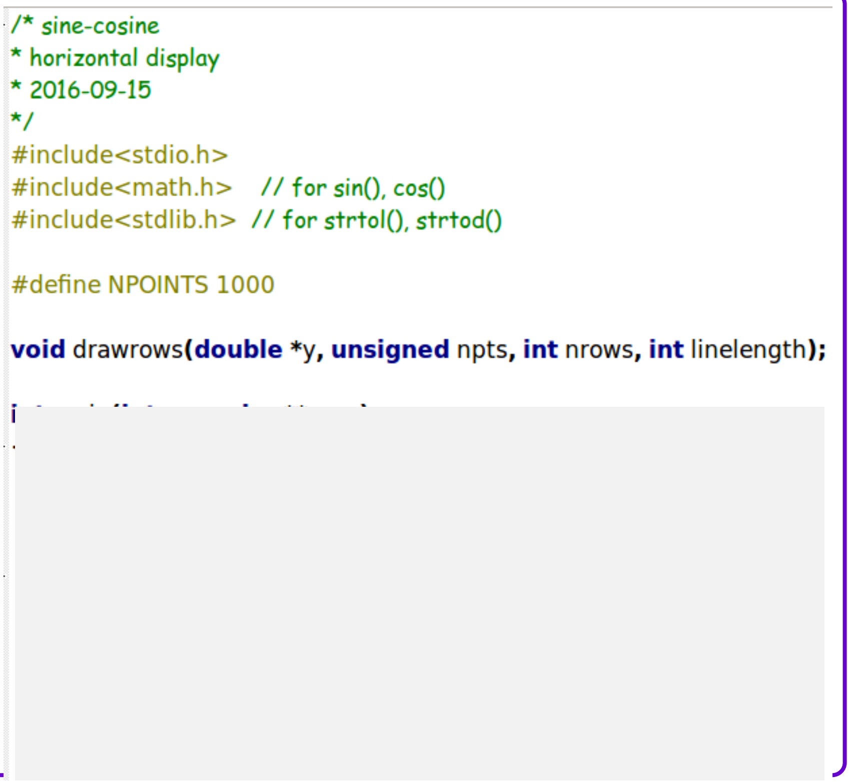


```c
/* vertical graph drawer */
#include <stdio.h>

void drawgraph(double *x, double *y, unsigned len, unsigned linelength)
{
    for (unsigned c = 0; c < linelength; c++)
        putchar('=');
    putchar('\n');

    for (unsigned i = 0; i < len; i++) {
        printf("%5.2lf| ", x[i]);
        unsigned halfline = (linelength - 7 - 6)/2;       // adjust for labels
        unsigned line = halfline + (halfline * y[i]);
        for (unsigned c = 0; c < line; c++)
            putchar('-');
        printf("% 5.2lf\n", y[i]);
    }

    for (unsigned c = 0; c < linelength; c++)
        putchar('=');
    putchar('\n');
}
```

Horizontal Sine-Cosine

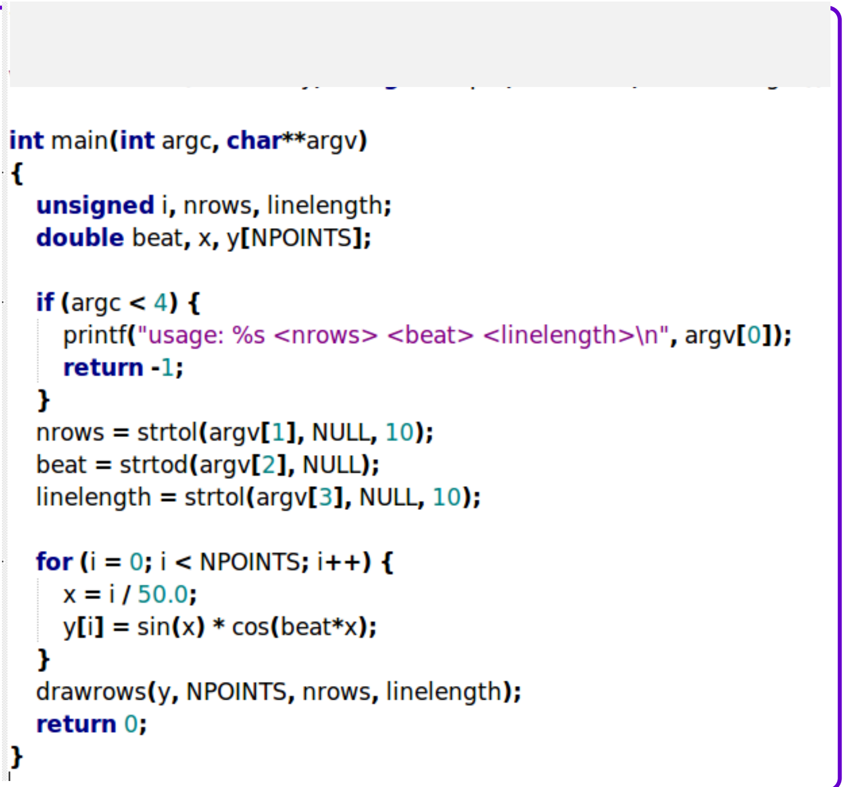
- Convert plot to more-standard horizontal display
- Each row of output reflects all y-values, so all y-values must be calculated (and saved) before any output
- Output function "drawrows()" determines character to print in each column based on corresponding y value

Horizontal sin-cosine - main() header material



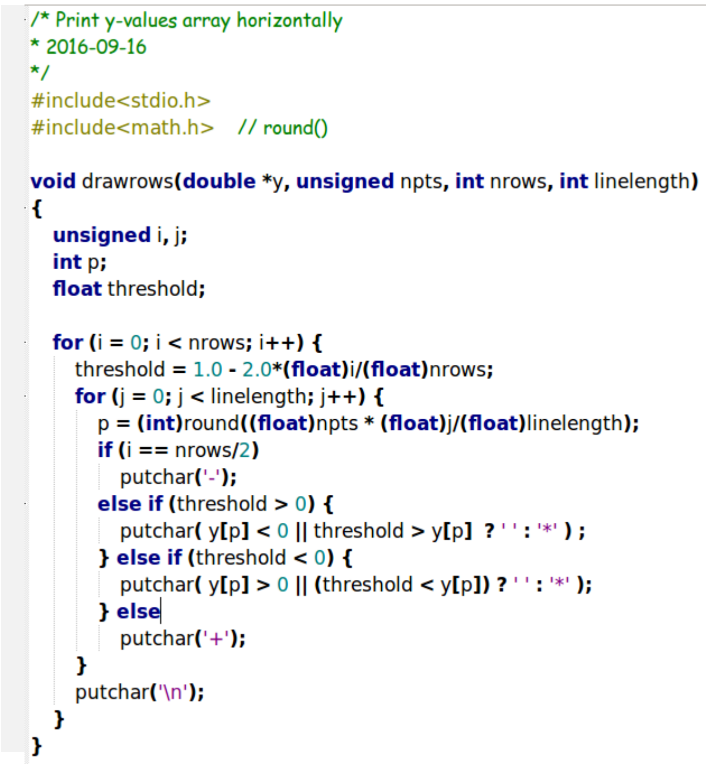
```
/* sine-cosine
* horizontal display
* 2016-09-15
*/
#include<stdio.h>
#include<math.h>    // for sin(), cos()
#include<stdlib.h>  // for strtol(), strtod()

#define NPOINTS 1000

void drawrows(double *y, unsigned npts, int nrows, int linelength);
```

Horizontal sin-cosine - main() body



```
int main(int argc, char**argv)
{
   unsigned i, nrows, linelength;
   double beat, x, y[NPOINTS];

   if (argc < 4) {
      printf("usage: %s <nrows> <beat> <linelength>\n", argv[0]);
      return -1;
   }
   nrows = strtol(argv[1], NULL, 10);
   beat = strtod(argv[2], NULL);
   linelength = strtol(argv[3], NULL, 10);

   for (i = 0; i < NPOINTS; i++) {
      x = i / 50.0;
      y[i] = sin(x) * cos(beat*x);
   }
   drawrows(y, NPOINTS, nrows, linelength);
   return 0;
}
```

Horizontal sin-cosine - drawrows()



```
/* Print y-values array horizontally
* 2016-09-16
*/
#include<stdio.h>
#include<math.h>    // round()

void drawrows(double *y, unsigned npts, int nrows, int linelength)
{
   unsigned i, j;
   int p;
   float threshold;

   for (i = 0; i < nrows; i++) {
      threshold = 1.0 - 2.0*(float)i/(float)nrows;
      for (j = 0; j < linelength; j++) {
         p = (int)round((float)npts * (float)j/(float)linelength);
         if (i == nrows/2)
            putchar('-');
         else if (threshold > 0) {
            putchar( y[p] < 0 || threshold > y[p]  ? ' ' : '*' ) ;
         } else if (threshold < 0) {
            putchar( y[p] > 0 || (threshold < y[p]) ? ' ' : '*' );
         } else
            putchar('+');
      }
      putchar('\n');
   }
}
```

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;

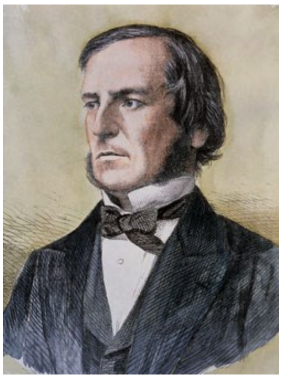
Some casts are necessary;
some casts aren’t really needed,
because the compiler can “do the
right thing” without any help.

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)

Why "Boolean"?

- George Boole, 1815 - 1864
  - English mathematician and logician
  - Introduced algebra of binary-valued systems, today called Boolean algebra



Boolean Operators

- Relational, *numeric* comparisons
  - Compare integers or floats to each other

==	is equal to	!=	is unequal to
<	is less than	>=	is greater or equal to
>	is greater than	<=	is less or equal to

- Boolean combinations
  - Operate on Boolean values, produce a Boolean

&&	AND	||	OR	!	NOT

  - Operators can combine Boolean values into bigger, more complex Boolean-valued expressions

Boolean Expressions – Usage Examples

- (7 > -5)
  - **True** (*a.k.a.* 1)
- (x == 0)
  - **True** if x is zero
  - *Not the same as* (x = 0) – be careful!
- a = 99.5;
  b = 1e4;
  c = (b < a);
  - c gets **False** (*a.k.a.* **0**)
- (x >= 97 && x <= 122)
  - **True** if x is in the range 97..122
    - These are ASCII lowercase…
- !(a == 0 || b == 0)
  - **True** if a and b are both non-zero
- z = (c && !d);
  - z gets **True** if c is **True** and d is **False**

Assignment Operators

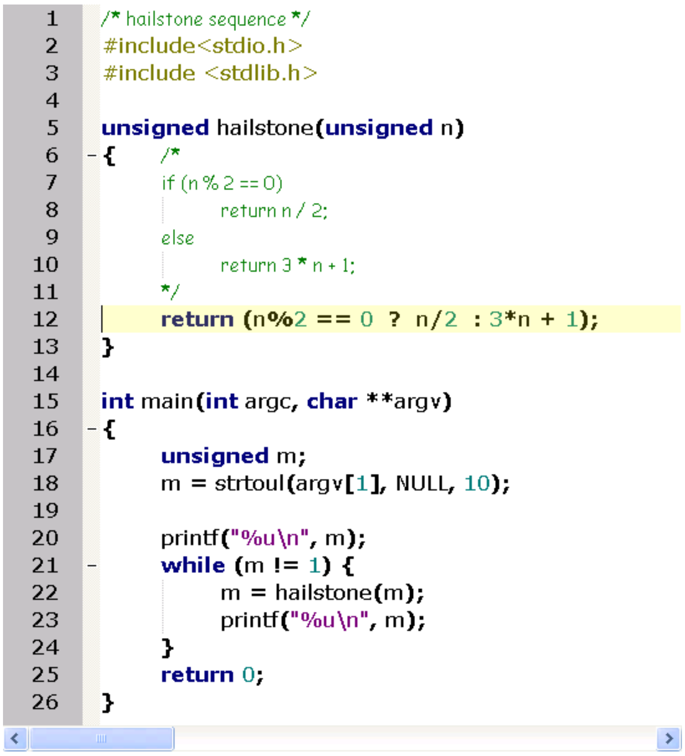
- Assignment in C is an operation that returns a value (like any other operation), and has the side-effect of changing the value of its left-hand operand.
- Available operators
  - assign:
    =
  - arithmetic-assign:
    +=   -=   *=   /=   %=
    &=   ^= |=   <<=   >>=

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?



```
/* 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 ;

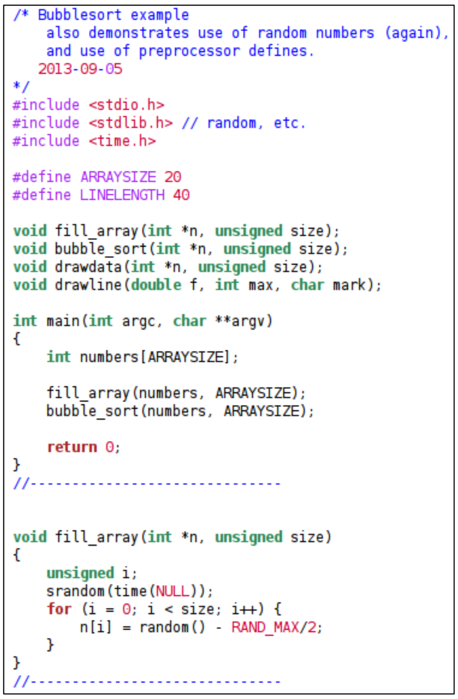
Example: Bubble Sort

- 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.

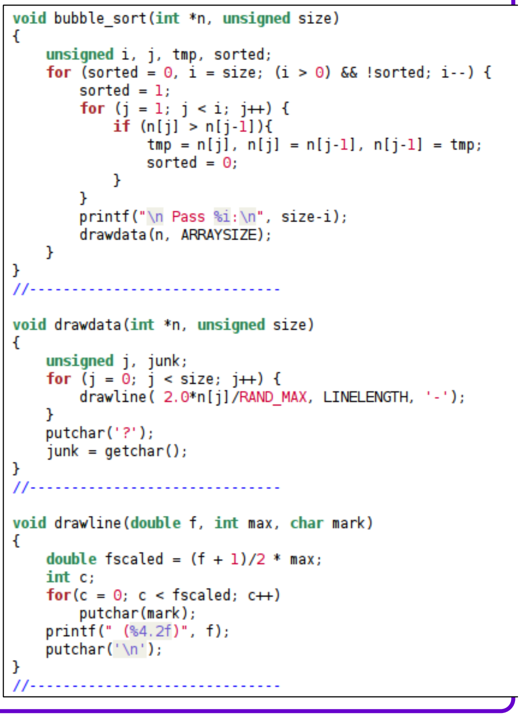
Bubble Sort - details

- 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 rowlength
- 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.
   2013-09-05
*/
#include <stdio.h>
#include <stdlib.h> // random, etc.
#include <time.h>

#define ARRAYSIZE 20
#define LINELENGTH 40

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

int main(int argc, char **argv)
{
    int numbers[ARRAYSIZE];

    fill_array(numbers, ARRAYSIZE);
    bubble_sort(numbers, ARRAYSIZE);

    return 0;
}
//-----------------------------


void fill_array(int *n, unsigned size)
{
    unsigned i;
    srandom(time(NULL));
    for (i = 0; i < size; i++) {
        n[i] = random() - RAND_MAX/2;
    }
}
//-----------------------------
```



```c
void bubble_sort(int *n, unsigned size)
{
    unsigned i, j, tmp, sorted;
    for (sorted = 0, i = size; (i > 0) && !sorted; i--) {
        sorted = 1;
        for (j = 1; j < i; j++) {
            if (n[j] > n[j-1]){
                tmp = n[j], n[j] = n[j-1], n[j-1] = tmp;
                sorted = 0;
            }
        }
        printf("\n Pass %i:\n", size-i);
        drawdata(n, ARRAYSIZE);
    }
}
//-----------------------------

void drawdata(int *n, unsigned size)
{
    unsigned j, junk;
    for (j = 0; j < size; j++) {
        drawline( 2.0*n[j]/RAND_MAX, LINELENGTH, '-');
    }
    putchar('?');
    junk = getchar();
}
//-----------------------------

void drawline(double f, int max, char mark)
{
    double fscaled = (f + 1)/2 * max;
    int c;
    for(c = 0; c < fscaled; c++)
        putchar(mark);
    printf(" (%4.2f)", f);
    putchar('\n');
}
//-----------------------------
```