Smith College

Computer Science

CSC352

Week #7 — Spring 2017

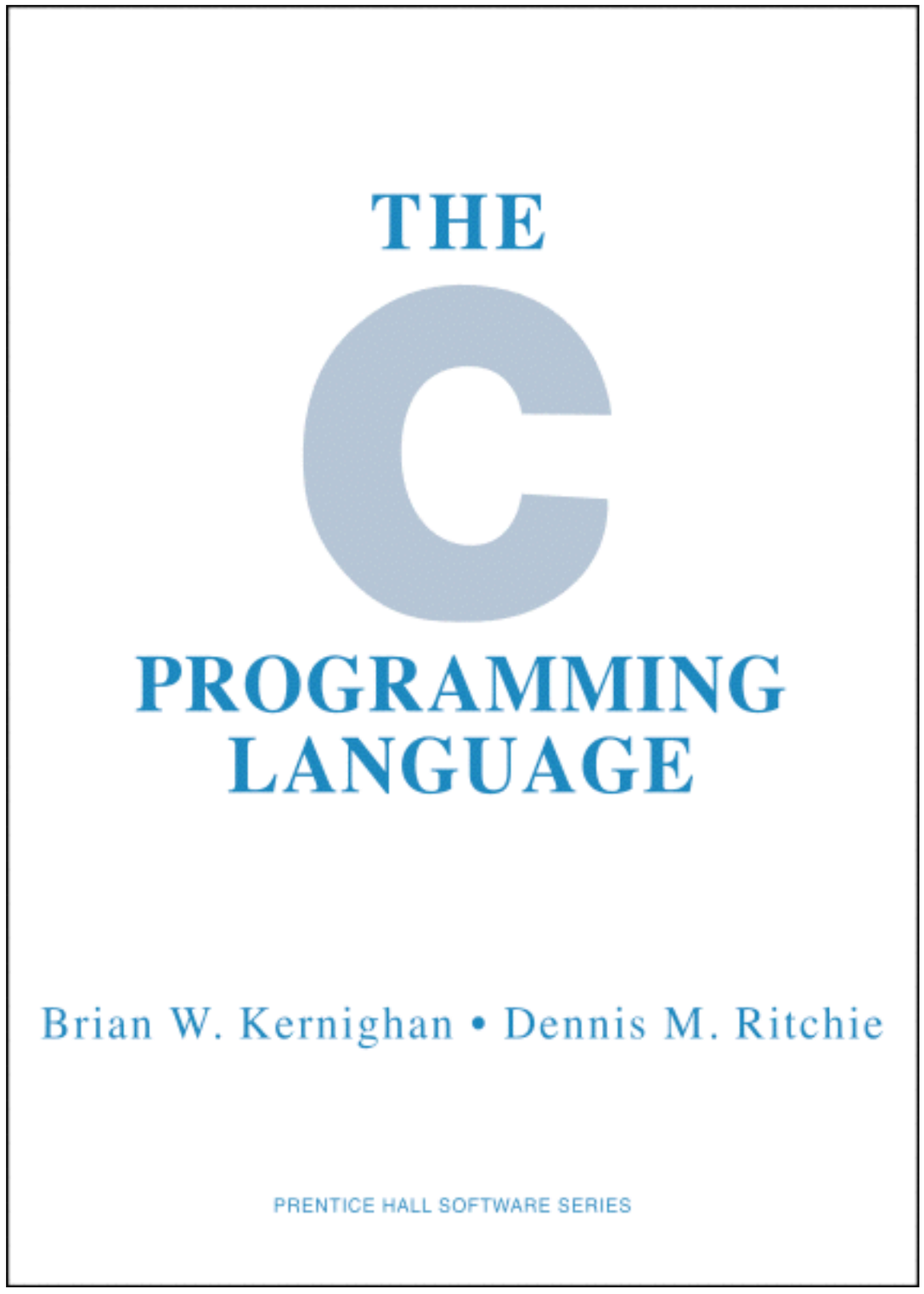
Introduction to C

Dominique Thiébaut

dthiebaut@smith.edu

Learning C in 2 Hours

D. Thiebaut



- Dennis Ritchie
- 1969 to 1973
- AT&T Bell Labs
- Close to Assembly
- Unix

- Standard
- Many languages based on C. (C++, Obj. C, C#)
- Many influenced by C (Java, Python, Perl)

C Lacks…

- Exceptions
- Garbage collection
- OOP
- Polymorphism
- But…

C Lacks…

- Exceptions
- Garbage collection
- OOP
- Polymorphism
- But… it is usually faster!

Good Reference

- Essential C, by Nick Parlante, Stanford U. http://cslibrary.stanford.edu/101/EssentialC.pdf

Hello World!

- Library
- Strings
- Block-structured language
- main()

```
#include <stdio.h>

void main() {
  printf("\nHello World\n");
}
```

Compiling on Aurora

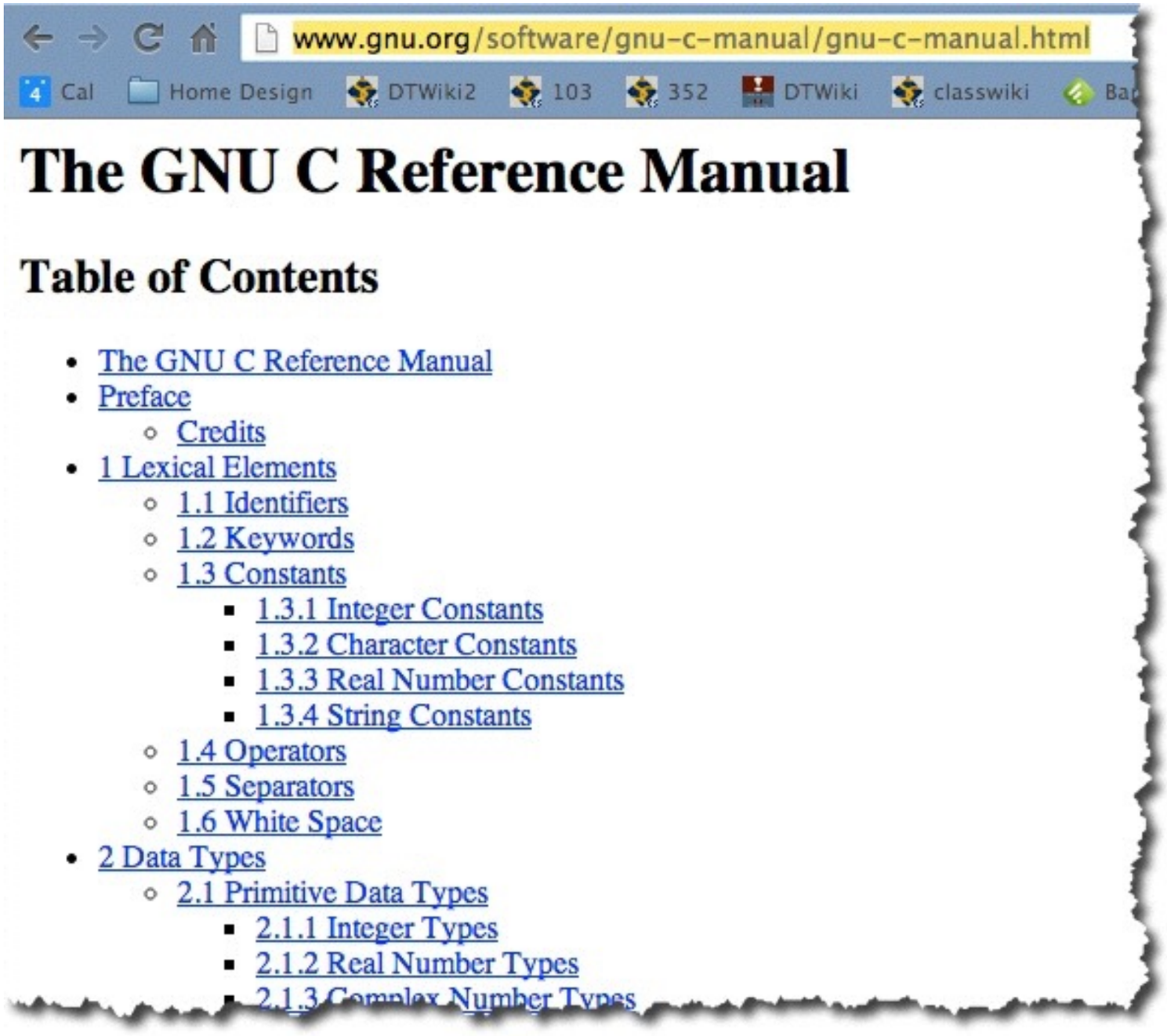
- gcc Gnu compiler
- `man gcc` for help

```
[~/handout]$ gcc hello.c
[~/handout]$ ./a.out

Hello World
```

```
[~/handout]$ gcc -o hello hello.c
[~/handout]$ ./hello

Hello World
```

Files

```
[~/handout]$ ls -l
total 28
-rwx------ 1 352a 352a 6583 Oct  6 16:41 a.out*
-rwx------ 1 352a 352a 6583 Oct  6 16:48 hello*
-rw------- 1 352a 352a   66 Oct  6 16:41 hello.c
-rw------- 1 352a 352a   67 Oct  6 16:39 hello.c~
```



Good Reference on C *Compiler*

- http://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html

Printing

- `printf( “string with %-operators”, list of vars);`
  - `%d int`
  - `%f float`
  - `%s string`

Variables

- Simple types
- No strings!
- No booleans (only 0 for *fals*e and !0 for *true*)
- No classes, no objects!

```
int    -> integer variable
short  -> short integer
long   -> long integer
float  -> single precision real (floating point) variable
double -> double precision real (floating point) variable
char   -> character variable (single byte)
```

Comments

```
/*
programName.c
author

This is the header
*/
#include <stdio.h>
#include <string.h>

void main() {
    // line comment
    print( “Hello!\n" )
}
```

Strings

```
#include <stdio.h>
#include <string.h>

void main() {
  char hello[] = "hello";
  char world[] = "world!";
  char sentence[100] = "";

  strcpy( sentence, hello );  // sentence <- "hello"
  strcat( sentence, " " );    // sentence <- "hello "
  strcat( sentence, world );  // sentence <- "hello world!"

  printf( "sentence = %s\n", sentence );
}
```

```
[~/handout]$ gcc strings2.c
[~/handout]$ a.out

sentence = hello world!
[~/handout]
```

Strings end with '\0'

```
#include <stdio.h>
#include <string.h>

void main() {
  char sentence[100] = "Hello world!";

  printf( "sentence = %s\n", sentence );
  sentence[5] = '\0';
  printf( "sentence = %s\n", sentence );
}
```

```
~/handout]$ a.out
sentence = Hello world!
sentence = Hello
[~/handout]$
```

Exercise

```
#include <stdio.h>
#includd <string.h>

void main() {
  int a   = 3;
  int b   = 5;
  int c   = 0;
  char firstName[] = "your first name here";
  char lastName[] = "your last name here";
  char fullName[100];
  ...
}
```

- make the program store the sum of a and b into c, and then print your full name and the value in c. Also, make it output the number of characters in your full name (it must count the number of chars using a string function (strlen))

Solution

```
#include <stdio.h>
#include <string.h>

void main() {
  int a = 3;
  int b = 5;
  int c = 0;
  char firstName[] = "Mickey";
  char lastName[] = "Mouse";
  char fullName[100];

  c = a + b;
  strcpy( fullName, firstName );
  strcat( fullName, " " );
  strcat( fullName, lastName );

  printf( "\nc = %d  Full name = %s\n\n", c, fullName );
}
```

For-Loops

No "int" declaration!!!

```
#include <stdio.h>

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

   // compute the sum of all the numbers from 1 to 100
   for ( i=1; i<=100; i++ ) {
      sum += i;
   }

   printf( "\nsum = %d\n\n", sum );
}
```

While-Loops

```
#include <stdio.h>

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

   // compute the sum of all the numbers from 1 to 100
   i = 1;
   while ( i<=100 ) {
      sum += i;     // could have also used i++
      i += 1;
   }

   printf( "\nsum = %d\n\n", sum );
}
```

Infinite Loops

```
#include <stdio.h>

void main() {

  while ( 1 ) {
      printf( “hello!\n” );
  }
}
```

```
#include <stdio.h>

void main() {

  for ( ;; ) {
    printf( “hello!\n” );
  }
}
```

Exercise

- Write a program that writes your full name underlined (line of dashes below). The underline length must be computed and the number of characters equal to the number of characters in your full name.

  Hints: strlen() returns # of chars

```
man strlen
```

Symbolic Constants

```
#include <stdio.h>

#define NAME       "Mickey"
#define HEIGHT     5
#define YEARBORN   1928

void main() {
  printf( "%s is %d inches high, and was created in %d\n\n",
          NAME, HEIGHT, YEARBORN );
}
```

Conditionals

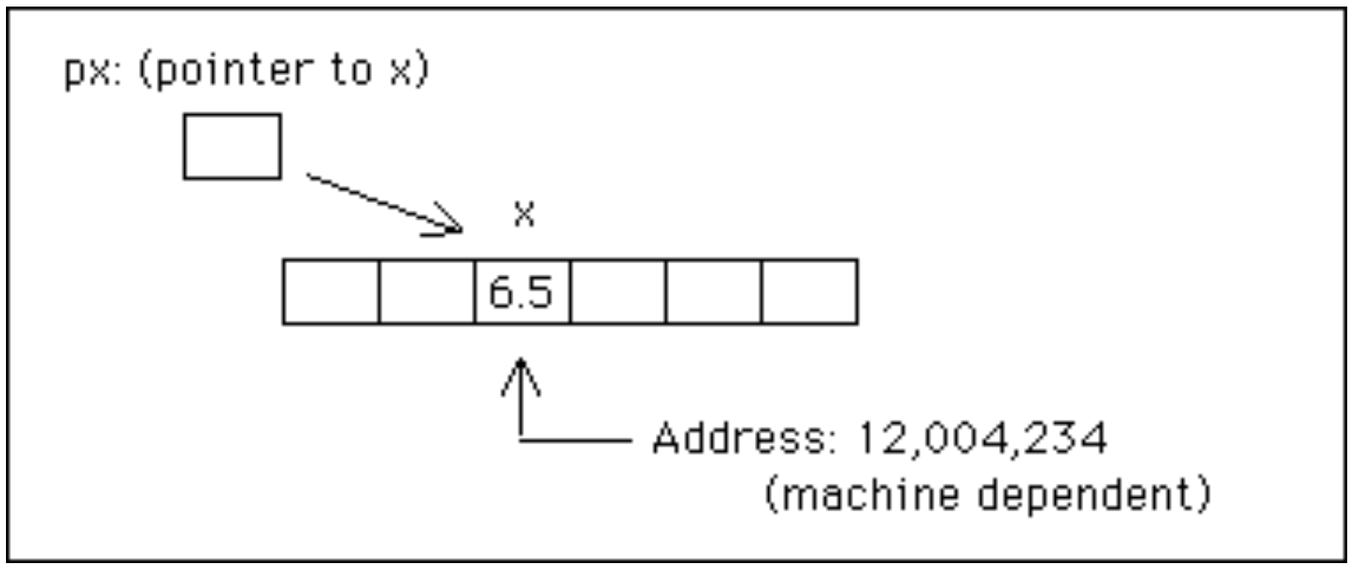
```
#include <stdio.h>

void main() {
   int a = 5;
   int b = 3;
   int c = 7;

   if ( a <= b && a <= c )
      printf( "%d is the smallest\n\n", a );
   else if ( b <= a && b <= c )
      printf( "%d is the smallest\n\n", b );
   else
      printf( "%d is the smallest\n\n", c );
}
```

Conditionals (cont'd)

```
switch ( ordinal_expression ) {
  case ordinal_value: {
    // ...
    break;
  }
  case ordinal_value: {
    // ...
    break;
  }
  default: {
    // ...
  }
}
```

Pointers

Concept



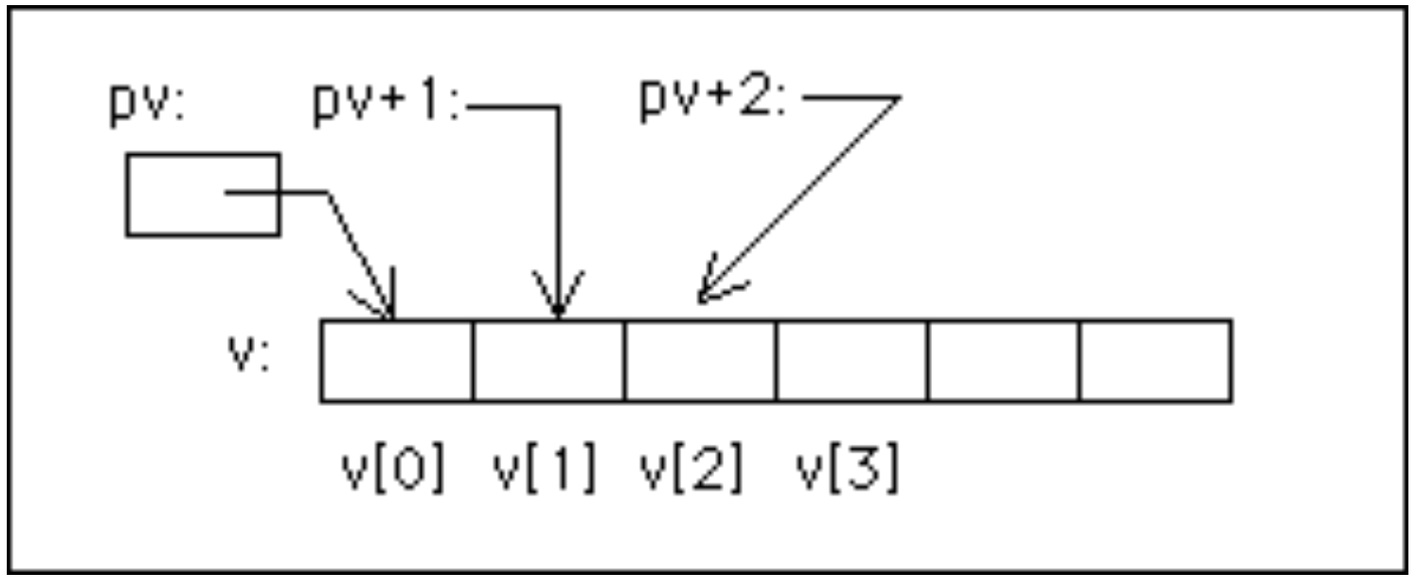

```
float x = 6.5;
float* px = &x;
```

Example: Initialize an Array

```
#include <stdio.h>
#define SIZE 10

int  main() {
    float A[SIZE];
    float* p;
    int i;

    p = A;
    for ( i=0; i<SIZE; i++ ) {
        *p = i;
        p++;
    }

    p = A;
    for ( i=0; i<SIZE; i++ ) {
        printf( "p=%p A[%d] = %1.2f *p = %1.2f\n",
                 p, i, A[i], *p );
        p = p + 1;
    }
}
```

```
a.out
p=0x7fff88d54560 A[0] = 0.00 *p = 0.00
p=0x7fff88d54564 A[1] = 1.00 *p = 1.00
p=0x7fff88d54568 A[2] = 2.00 *p = 2.00
p=0x7fff88d5456c A[3] = 3.00 *p = 3.00
p=0x7fff88d54570 A[4] = 4.00 *p = 4.00
p=0x7fff88d54574 A[5] = 5.00 *p = 5.00
p=0x7fff88d54578 A[6] = 6.00 *p = 6.00
p=0x7fff88d5457c A[7] = 7.00 *p = 7.00
p=0x7fff88d54580 A[8] = 8.00 *p = 8.00
p=0x7fff88d54584 A[9] = 9.00 *p = 9.00
```

Arrays




```
TYPE v[DIM];
TYPE* pv;

pv = v;
```

Arrays

- The name of an array is a pointer to the first cell of the array.

```
char name[DIM];
```

- `name` is the same as `&(name[0])`

* and &

- * has two meanings, depending on context
  - "Pointer to"
  - "Contents of"
- & means "the address of"

* and &

```
int A[DIM];
int* p = A;              // “int pointer p”
int *q = &A[0];          // “int pointer q”

*p = 3;                  // what p is pointing to gets 3
*(q+1) = 5;              // what q is pointing to gets 5
```

Functions

- Functions are always declared before they are used
- Functions can return values of simple types (int, char, floats), and *even* pointers.
- Functions get parameters of simple types, and pointers.
- Passing by value is automatic. Passing by reference requires passing a pointer.

Functions

```
#include <stdio.h>

int sum( int a, int b ) {
    return a+b;
}

void main() {
    int x = 10;
    int y = 20;
    int z;

    z = sum( x, y );
    printf( "z = %d\n", z );

    z = sum( 3, 8 );
    printf( "z = %d\n", z );

    printf( "sum( 11, 22) = %d\n", sum( 11, 22 ) );
}
```

```
z = 30
z = 11
sum( 11, 22) = 33
```

We stopped here last time…

Functions

(Incomplete code… Add missing elements!)

```
#include <stdio.h>

void sum2( int a, int b, int  c ) {
    c = a+b;
}

void main() {
    int x = 10;
    int y = 20;
    int z;

    sum2( x, y,   z );
    printf( "z = %d\n", z );

    sum2( 3, 8,   x );
    printf( "x = %d\n", x );
}
```

Pass by Reference!

```
z = 30
x = 11
```

Input

```
#include <stdio.h>

void main() {
    int age;
    float myPi;
    char name[80];

    printf( "Enter your name, please: " );
    fgets( name, sizeof(name) stdin );
                    // will truncate to first
                    // 80 chars entered

    printf( "Enter your age: " );
    scanf( "%d", &age );

    printf( "Enter your version of pi: " );
    scanf( "%f", &myPi );

    printf( "%s is %d years old, and thinks pi is %1.10f\n\n",
            name, age, myPi );
}
```

Input (cont'd)

```
a.out
Enter your name, please: Mickey
Enter your age: 21
Enter your version of pi: 3.14159
Mickey is 21 years old, and thinks pi is 3.1415901184
```

Exercise

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

#define   N 10

// functions go here...

void main() {
    int A[N] = { 3, 2, 1, 0, 6, 5, 9,  8, 7 };

    // your code goes here
}
```

- Write a C program (no functions) that finds the smallest, the largest, and computes the sum of all the ints in A.
- Write another program that does the same thing but uses functions. The results are passed using return statements

Exercise

- Write another program that does the same thing but uses functions, and this time the results are *passed* back *by reference*.

We Stopped Here Last Time

https://farm1.staticflickr.com/44/145211107_f2020ec804_z.jpg

Dynamic Variables

Malloc

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

void main() {
    int *A, N, i, smallest;

    printf( "How many ints? " );
    scanf( "%d", &N );
    A = (int *) malloc( N * sizeof( int ) );

    for ( i=0; i<N; i++ ) {
        printf( "> " );
        scanf( "%d", &(A[i]) );
    }

    smallest = A[0];
    for ( i=1; i<N; i++ )
        if ( A[i] < smallest )
            smallest = A[i];
    free( A );

    printf( "The smallest = %d\n", smallest );
}
```

Exercise

- Modify the previous program that computed the min and max of an array, but this time you allocate the array dynamically from the user input,
  i.e. ask the user for number of ints, then the ints.
  (use *scanf("%d",&x)* to get an int from keyboard into x)

c/malloc1.c

sizeof()

can be tricky…

sizeof ()

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

int main( int argc, char* argv[] ) {
  int A[] = { 1, 2, 3, 4, 5 };
  int *B = (int *) malloc( 5 * sizeof( int ) );
  int *p = A;
  char name[] = "Smith College";
  int a = 3;
  float x = 3.14159;
  int i;

  for ( i=0; i<5; i++ ) B[i] = i;

  printf( "sizeof(A)    = %lu\n", sizeof( A ) );
  printf( "sizeof(A[0]) = %lu\n", sizeof( A[0] ) );
  printf( "sizeof(B)    = %lu\n", sizeof( B ) );
  printf( "sizeof(B[0]) = %lu\n", sizeof( B[0] ) );
  printf( "sizeof(p)    = %lu\n", sizeof( p ) );
  printf( "sizeof(*p)   = %lu\n", sizeof( *p ) );
  printf( "sizeof(name) = %lu\n", sizeof( name ) );
  printf( "strlen(name) = %lu\n", strlen( name ) );
  printf( "sizeof(a)    = %lu\n", sizeof( a ) );
  printf( "sizeof(x)    = %lu\n", sizeof( x ) );

  return 0;
}
```

sizeof ()

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

int main( int argc, char* argv[] ) {
  int A[] = { 1, 2, 3, 4, 5 };
  int *B = (int *) malloc( 5 * sizeof( int ) );
  int *p = A;
  char name[] = "Smith College";
  int a = 3;
  float x = 3.14159;
  int i;

  for ( i=0; i<5; i++ ) B[i] = i;

  printf( "sizeof(A)    = %lu\n", sizeof( A ) );
  printf( "sizeof(A[0]) = %lu\n", sizeof( A[0] ) );
  printf( "sizeof(B)    = %lu\n", sizeof( B ) );
  printf( "sizeof(B[0]) = %lu\n", sizeof( B[0] ) );
  printf( "sizeof(p)    = %lu\n", sizeof( p ) );
  printf( "sizeof(*p)   = %lu\n", sizeof( *p ) );
  printf( "sizeof(name) = %lu\n", sizeof( name ) );
  printf( "strlen(name) = %lu\n", strlen( name ) );
  printf( "sizeof(a)    = %lu\n", sizeof( a ) );
  printf( "sizeof(x)    = %lu\n", sizeof( x ) );

  return 0;
}
```

```
sizeof(A)    = 20
sizeof(A[0]) = 4
sizeof(B)    = 8
sizeof(B[0]) = 4
sizeof(p)    = 8
sizeof(*p)   = 4
sizeof(name) = 14
strlen(name) = 13
sizeof(a)    = 4
sizeof(x)    = 4
```

sizeof ()

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

int main( int argc, char* argv[] ) {
  int A[] = { 1, 2, 3, 4, 5 };
  int *B = (int *) malloc( 5 * sizeof( int ) );
  int *p = A;
  char name[] = "Smith College";
  int a = 3;
  float x = 3.14159;
  int i;

  for ( i=0; i<5; i++ ) B[i] = i;

  printf( "sizeof(A)    = %lu\n", sizeof( A ) );
  printf( "sizeof(A[0]) = %lu\n", sizeof( A[0] ) );
  printf( "sizeof(B)    = %lu\n", sizeof( B ) );
  printf( "sizeof(B[0]) = %lu\n", sizeof( B[0] ) );
  printf( "sizeof(p)    = %lu\n", sizeof( p ) );
  printf( "sizeof(*p)   = %lu\n", sizeof( *p ) );
  printf( "sizeof(name) = %lu\n", sizeof( name ) );
  printf( "strlen(name) = %lu\n", strlen( name ) );
  printf( "sizeof(a)    = %lu\n", sizeof( a ) );
  printf( "sizeof(x)    = %lu\n", sizeof( x ) );

  return 0;
}
```

Different!

```
sizeof(A)    = 20
sizeof(A[0]) = 4
sizeof(B)    = 8
sizeof(B[0]) = 4
sizeof(p)    = 8
sizeof(*p)   = 4
sizeof(name) = 14
strlen(name) = 13
sizeof(a)    = 4
sizeof(x)    = 4
```

File I/O output

```
#include <stdio.h>

void main() {
    FILE *fp;
    int i;
    char name[] = "Smith College";

    fp = fopen("hello.txt", "w");  // open file for writing

    fprintf(fp, "\nHello " );      // write constant string
    fprintf(fp, "%s\n\n", name );  // write string

    fclose(fp);                    // close file
}
```

File I/O: Reading Text

```
#include <stdio.h>

void main() {
    FILE *fp;
    char line[80];

    fp = fopen( "fgets2.c", "r" );  // open file for reading

    while ( !feof( fp ) ) {         // while not eof
        fgets( line, 80, fp );      // get at most 80 chars
        if ( feof( fp ) )           // if eof reached stop
            break;
        line[79] = '\0';            // truncate line to be safe
        printf( "%s", line );       // print it
    }

    close( fp );                    // close file
}
```

c/readFile.c

File I/O: Input Numbers

```
[~handout] cat fileOfInts.txt
4
1  2  3
4  5  6
7  8  9
10  11  12
```

File I/O: Reading Ints

```
#include <stdio.h>

void main() {
    FILE *fp;
    char line[80];
    int N, n1, n2, n3;

    fp = fopen( "fileOfInts.txt", "r" ); // 1st number is # of lines
                                         // then 3 ints per line

    if ( feof( fp ) ) {
        printf( "Empty file!\n\n" );
        return;
    }

    // get the number of lines
    fscanf( fp, "%d", &N );

    while ( !feof( fp ) ) {
        fscanf( fp, "%d %d %d", &n1, &n2, &n3 );
        if ( feof( fp ) )
            break;
        printf( "%d, %d, %d\n", n1, n2, n3 );
    }

    close( fp );
}
```

c/readFileNumbers.c

File I/O: Reading Ints

```
[~handout] a.out
1, 2, 3
4, 5, 6
7, 8, 9
10, 11, 12
```

Exercise

- At the Bash prompt type:

```
cat > data.txt
4
10
1
2
3
^D
```

- Write a program that reads the file data.txt, which has the number of ints in contains on the first line, then one int per line for the rest of the file. Your program must use a dynamically created array to store the numbers, and find the min and max of the array, and print them.