Introduction

Fall Semester

Topic Outline

- Programming in C
  - Pointers
  - Input-Output
- Embarrassingly Parallel
  - Message Passing Interface
  - Projectile motion
  - Fractal generation
  - Cellular Automata
  - OpenGL graphics
- Coupled Systems
  - Heat Equation
  - Nearest neighbor
  - N-Body problem
  - Round robin
- Additional Tools
  - XMT and OpenMP
  - Threads and CUDA
  - Sockets

The C Programming Language:

"C is quirky, flawed, and an enormous success."
–Dennis Ritchie

"C makes it easy to shoot yourself in the foot; C++ makes it harder, but when you do, it blows away your whole leg."
–Bjarne Stroustrup

"Some people...think that C is a real programming language, but they are sadly mistaken... almost-portable assembly language."
–Linus Torvalds

Hello World[1]

```
#include <stdio.h>              // comments

int main(void)                  // entry point of program
{
   printf("hello, world\n");
   return 0;                    // successful completion
}
```

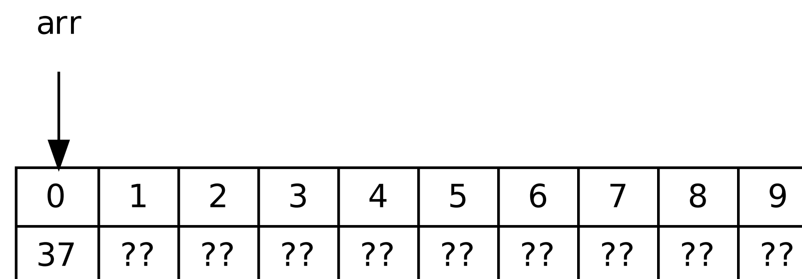
[1] Kernighan, Brian, and Dennis Ritchie. The C Programming Language. 1st ed. Englewood Cliffs, NJ: Prentice Hall, 1978.

Pointers (1)

```
#include <stdio.h>
int main(int argc,
         char* argv[])    // String[] args in Java
{
   int arr[10];           // array of ten integers
   arr[0]=37;             // data is uninitialized
   printf("%d\n",*arr);   // pointer dereference
   return 0;              // output is thirty-seven
}
```

Pointers (2)

- Array variable names are nothing more than pointers to the first element stored in a contiguously allocated block of memory.
- A given slot can be accessed using pointer arithmetic; this is why the first element has index zero. The mystery is solved!
- Syntax: `*arr = *(arr+0) = arr[0] = 37;`



Pointers (3)

```
void disp(int* a,int n)   // pass-by-value, but...
{                         // copy the pointer only
   int k;                 // so data is not copied
   for(k=0;k<n;k++)
      printf("%d\n",a[k]);
}

main:
   disp(arr,10);          // note, no arr.length
```

Pointers (4)

```
int a[10],b[10],*c,k;       // arrays, pointer, int
...
for(k=0;k<10;k++)
   *(b+k)=*(a+k);           // pointer arithmetic

c[0]=12345;                 // uh-oh, need malloc
c=a;                        // only copies pointer
for(k=0;k<10;k++)
   printf("%d\n",*c++);     // tricky plus-plus
```

Input

```
char ch;                    // not a pointer
while(1)                    // infinite loop
{
   ch=getchar();            // simple input
   if(ch==EOF)              // CTRL-D is EOF
      break;

   printf("***%c***\n",ch);
}
```

Output

```
char* str="hello";          // array of characters
char tmp[6];                // extra slot for ‘\0’

printf("%s\n",str);         // print entire string
printf("%c\n",str[0]);      // print first character

sprintf(tmp,"%s",str);      // print onto buffer
printf("%c\n",tmp[5]);      // output is a blank
printf("%d\n",tmp[5]);      // output is zero
```

American Standard Code for Information Interchange (ASCII)

- One byte only so `sizeof(char)==1` in C.

char	int	binary	char	int	binary
'\0'	0	0000 0000	'A'	65	0100 0001
'\n'	10	0000 1010	'B'	66	0100 0010
' '	32	0010 0000	'C'	67	0100 0011
'0'	48	0011 0000	'Z'	90	0101 1010
'1'	49	0011 0001	'a'	97	0110 0001
'2'	50	0011 0010	'b'	98	0110 0010
'3'	51	0011 0011	'c'	99	0110 0011
'9'	57	0011 1001	'z'	122	0111 1010

Lab Assignment: Spanning Plot

- Download, compile, run, and understand the `demo.c` file.
- Use floodfill to label each connected component:
  - 1, 2, 3, ... , 9, A, B, C, ... , Z, a, b, c, ... , z, *
- Determine if a component spans the entire grid. If so, which?
  - To span it must reach all four borders: top, bottom, left, right
- Run 10,000 trials and report the percent where spanning exists.
- Loop p from 0.0 to 1.0 in steps of 0.01 and report % for each.
- Use `gnuplot` to display the results, then increase the grid size from 40×30 and re-plot. What happens as the size increases?