

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 World¹

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

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

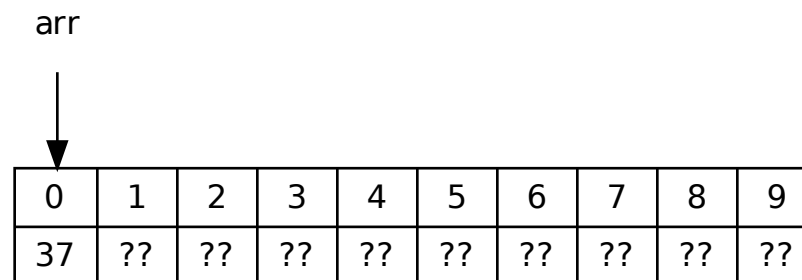
¹Kernighan, Brian, and Dennis Ritchie. The C Programming Language. 1st ed. Englewood Cliffs, NJ: Prentice Hall, 1978.

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

Example Program: Miscellaneous

- `double`, `math`, `sqrt`
- `string`, `strlen`
- `printf`, `%f`
- Compile
 - `gcc -lm miscellaneous_0.c`
 - Or, `gcc -lm -o misc miscellaneous_0.c`
- Run
 - `./a.out`, or `./misc`
 - Command-line file redirection with `<`, `>`, `>>`, and `2>`.

Lab Assignment: Entropy Calculation

- Input a message as a `char*` and call its length n .
- Calculate the entropy of each symbol x where:
 - The number of times x appears in the message is F_x .
 - The probability of drawing x at random is $P_x = F_x/n$.
 - The entropy of x is $H_x = -P_x \cdot \log_2(P_x)$.
- Calculate the total entropy H by summing all the H_x .
- Calculate the number of redundant bits in 8-bit ASCII.
 - The theoretical minimum number of bits is $\lceil H \cdot n \rceil$.
- See also Null and Lobur pages 310-311.

Example Entropy Calculation (1)

x	F_x	P_x	H_x	bits
=====				=====
h	1	0.100	0.332	3.322
e	1	0.100	0.332	3.322
l	3	0.300	0.521	1.737
o	2	0.200	0.464	2.322
w	1	0.100	0.332	3.322
r	1	0.100	0.332	3.322
d	1	0.100	0.332	3.322

- Start simple and only once everything is working then try a more complicated input like the `declaration.eng` file.

Example Entropy Calculation (2)

- The bits column, not required by the lab, indicates the theoretical minimum number of bits for each symbol in the message.
- The entropy H_x multiplies the bits times P_x , essentially weighting the value so that the overall H gives the average number of bits needed across all symbols. Thus, $\lceil H \cdot n \rceil$ is the total bits.
- Encoded, 011010000001001011011000111 contains 27 bits for a compression factor of $1 - 27/80 = 66.25\%$, plus overhead:

Char	Code	Char	Code	Char	Code	Char	Code
h	011	l	00	w	0101	d	111
e	0100	o	10	r	110		

Huffman Compression

- The big idea is to replace ASCII with a custom encoding scheme.
- Symbols that appear more often are given shorter codewords and symbols that appear less often are given longer codewords.
- Care must be taken that the encoded message can be uniquely decoded. For instance, we can't have a scheme where one codeword is `001` and another is `0010110`. How would we know when to stop decoding the symbol?
- There is an overhead cost to represent the custom scheme that is not required for ASCII since it is an established standard.

Example Custom Scheme

Symbol	Codeword	Symbol	Codeword
-	01	O	10100
T	000	R	10101
L	0010	A	11011
Y	0011	U	110100
I	1001	N	110101
H	1011	F	1000100
P	1100	M	1000101
E	1110	C	1000110
G	1111	D	1000111
S	10000		

Corresponding Message

```
1011100111111111001011100000011011100100111111111001011
1000000110111001010011000100010111110011000111101001111
0110111101110000011110110110001110110101010001011111001
1000101101001100010001011111001110010011111100000110011
10101011101101101111101001010110101001101000101111100110
0011011011000100000110011101010111011011000100001011010
0101011010100110110111001111111110010111000000110111001
0011111111100101110000001101110010100110001
```

- Note. If these “bits” are stored in ASCII then ‘0’ is 00110000 and ‘1’ is 00110001 and we only “compress” the message in theory. Or, use bit operators to read and write a binary file.

Start of the Decode Process

- The first eight bits of the message are: 10111001
- Since 1011 is the codeword for ‘H’ and 1001 is ‘I’ the decoded message begins with “HI”. A friendly start!
- Checking the custom scheme shows there is no ambiguity as to which symbols could be represented by this bit sequence.
- Representing these two symbols in ASCII would require twice the number of bits, hence compression.

Lab Assignment: Huffman Decode

- Input the number of symbols.
 - Study how this is done in the `decode_SHELL.c` file.
- Input the custom scheme and the message.
- Decode the message.
 - There is a guaranteed unique way to do this.
- Output the decoded message.
 - Start with `decode.txt` which contains the example scheme and message included in these slides.
 - Additional examples are provided on the course website.