

Pointers

- int *ptr = a;
- ptr[i] \rightarrow *(a+i)
- What is wrong with this:
- *ptr = (int*)malloc(sizeof(int)*10);

Practical example

- Say I am going to take everyone's age in the room (example 30 students)....will input one at a time, and want a sorted list all the time
- How would your c program look like ?

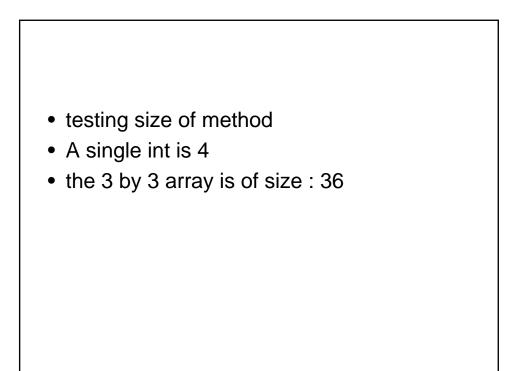
array

- Create the n size array
- Get number
- Figure where it goes
- Move everyone over to make place

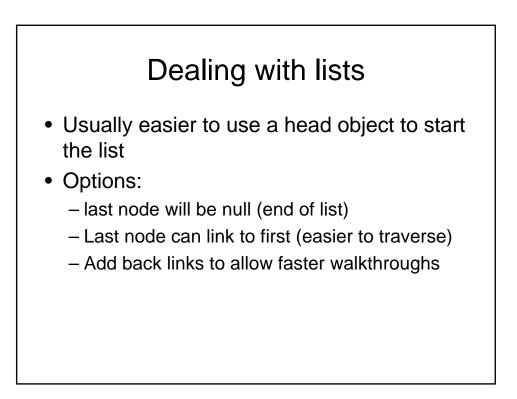
pointer

• How to do it with pointers?

Quick question: Output?	
#include <stdio.h> #include<stdlib.h></stdlib.h></stdio.h>	
int main(){ typedef int mag[3][3];	
printf("testing size of method\n");	
printf("A single int is %d\n",sizeof(int));	
printf("the 3 by 3 array is of size : %d\n",sizeof(mag));	
return 0;	
}	



```
• First we define:
struct ELEMENT {int value; struct ELEMENT
 *next; };
struct ELEMENT list;
list.next = (struct
 ELEMENT*)malloc(sizeof(struct ELEMENT));
• list.value = 20;
(*list.next).value = 22;
printf("val is %d\n",list.next->value);
```



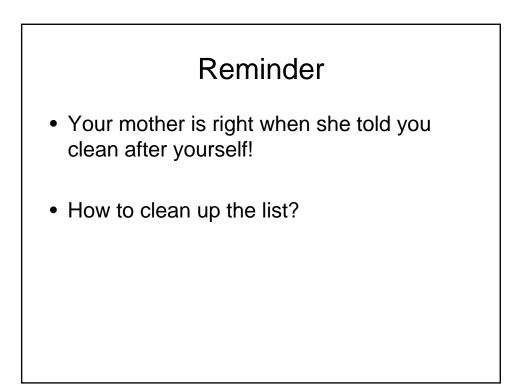
compare

- So if I have 10 items
- Whats the difference between an array and linked list?

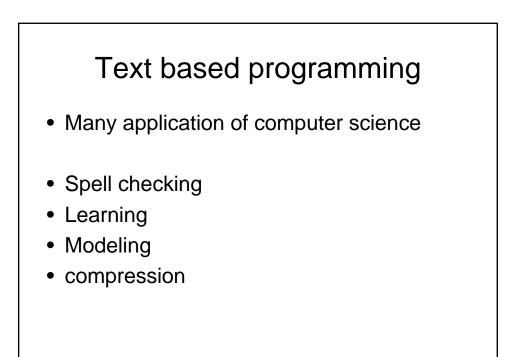
Working with lists

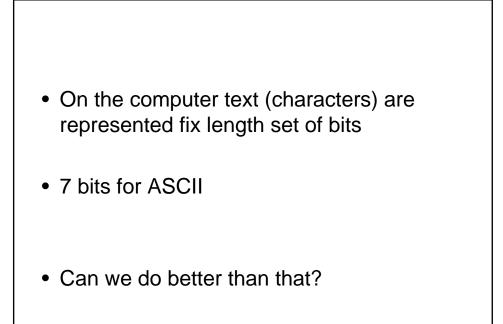
```
void add(llnode **head, int data_in) {
  llnode *tmp;
  if ((tmp = malloc(sizeof(*tmp))) == NULL){
  ERR_MSG(malloc);
  void)exit(EXIT_FAILURE);
  }
  tmp->value = data_in;
  tmp->next = *head;
  *head = tmp;
  }
```

```
/* ... inside some function ... */
llnode *head = NULL;
.....
add(&head, some_data);
```



```
void freelist(llnode *head) {
llnode *tmp;
while (head != NULL) {
free(head->data);
tmp = head->next;
free(head); head = tmp;
}
```





Compression

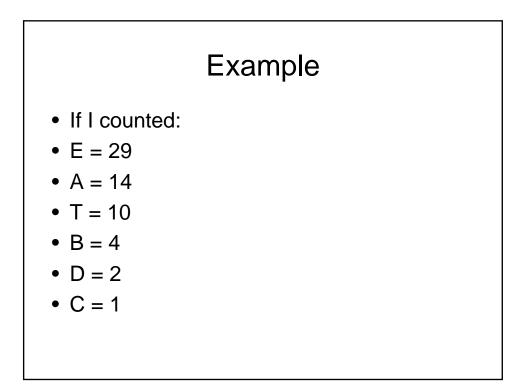
• If we can use less bits for higher occurring characters, overall we will use less bits in our text file

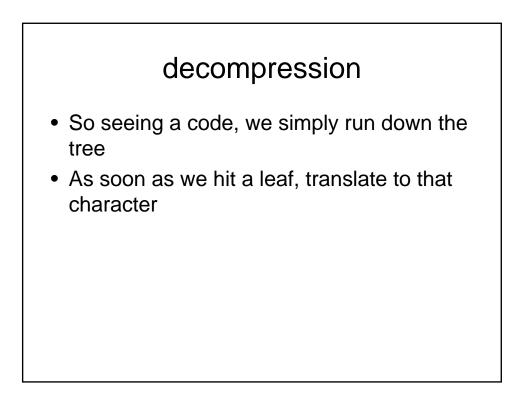
Binary tree

- Let me introduce a data structure to you
- A binary tree has a node with optional left and right children
- Think of it as a linked list with two links

Hoffman compression

- 1. Create a frequency count of each of your characters in your file
- 2. Start to build a binary tree always combining 2 lowest frequencies into one tree the resulting frequency is the combined frequencies
- 3. Going left is 0, going right is 1





Compressing text

How would you use huffman to compress text??

File manipulations

• FILE *fopen (const char *path, const char *mode);

FILE *Fp;
Fp = fopen("/home/johndoe/input.dat", "r");
fscanf(Fp, "%d", &x);
fprintf(Fp, "%s\n", "File Streams are cool!");
int fclose(FILE *stream);

Command line arguments

- Many times you want to pass in specific information to your program as command line args
- Tool for helping you do this:

```
int getopt(int argc, char * const argv[], const char
 *optstring);
extern char *optarg;
extern int optind, opterr, optopt;
```

Change main method

• int main(int argc, char **argv)

• ./junk -b something data.txt

```
int ich;
  while ((ich = getopt (argc, argv, "ab:c")) != EOF) {
   switch (ich) {
     case 'a': /* Flags/Code when -a is specified */
       break;
      case 'b': /* Flags/Code when -b is specified */
               /* The argument passed in with b is specified */
               /* by optarg */
       break;
      case 'c': /* Flags/Code when -c is specified */
       break;
     default: /* Code when there are no parameters */
       break;
   }
  }
 if (optind < argc) {
   printf ("non-option ARGV-elements: ");
   while (optind < argc)</pre>
     printf ("%s ", argv[optind++]);
   printf ("\n");
  }
```

Shift Gears

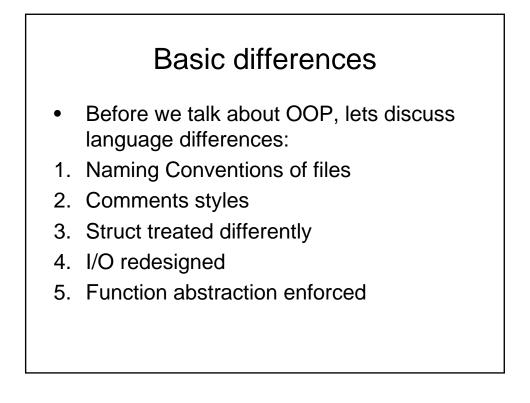
- Hopefully you feel comfortable looking at c and working in c.
- Some background:
 - Why are we covering all these languages so quickly?
 - What are you supposed to be taking out of the course?
 - How does c++ fit into this?
 - Bottom line
- Intro to c++

differences between c++ and c

- history and background
- object-oriented programming with classes
- very brief history...
 - C was developed 69-73 at Bell labs.
 - C++ designed by Bjarne Stroustrop at AT&T Bell Labs in the early 1980's
 - originally developed as "C with classes"
 - Idea was to create reusable code
 - development period: 1985-1991
 - ANSI standard C++ released in 1991

Four main OOP concepts

- abstraction
 - $-\,$ creation of well-defined interface for an object, separate from its implementation
 - e.g., Vector in Java
 - e.g., key functionalities (init, add, delete, count, print) which can be called independently of knowing how an object is implemented
- encapsulation
 - keeping implementation details "private", i.e., inside the implementation
- hierarchy
 - an object is defined in terms of other objects
 - Composition => larger objects out of smaller ones
 - Inheritance => properties of smaller objects are "inherited" by larger objects
- polymorphism
 - use code "transparently" for all types of same class of object
 - i.e., "morph" one object into another object within same hierarchy



Hello.cpp

```
#include <iostream.h>
#include <stdio.h>
main() {
  cout << "hello world\n";
  cout << "hello" << " world" << "\n";
  printf( "hello yet again!\n" );
  }
• compile using:
g++ hello.cpp -o hello</pre>
```

• like gcc (default output file is a.out)

