arrays and strings: pointers and memory allocation

- Why not rely solely on string and Vector classes?
  - how are string and Vector implemented?
  - lower level access can be more efficient (but be leery of claims that C-style arrays/strings *required* for efficiency)
  - real understanding comes when more levels of abstraction are understood

- string and vector classes insulate programmers from inadvertent attempts to access memory that’s not accessible
  - what is the value of a pointer?
  - what is a segmentation violation?
Contiguous chunks of memory

- In C++ allocate using array form of new
  ```cpp
  int * a = new int[100];
  double * b = new double[300];
  ```

- new [] returns a pointer to a block of memory
  ➤ how big? where?

- size of chunk can be set at runtime, not the case with
  ```cpp
  int a[100];
  cin >> howBig;
  int a[howBig];
  ```

- delete [] a; // storage returned

```cpp
int * a = new int[100];
```
In C, malloc is used to allocate memory

```c
int * a = (int *) malloc(100 * sizeof(int));
double * d = (double *) malloc(200 * sizeof(double));
```

- malloc must be cast, is NOT type-safe (returns void *)
  - void * is ‘generic’ type, can be cast to any pointer type

- free(d); // return storage

```c
int * a = (int *) malloc(100*sizeof(int));
```
Address calculations, what is sizeof(...)?

- x is a pointer, what is x+33?
  - a pointer, but where?
  - what does calculation depend on?

- result of adding an int to a pointer depends on size of object pointed to

- result of subtracting two pointers is an int:

\[(d + 3) - d == \underline{\text{_______}}\]

\[
\begin{array}{cccc}
0 & 1 & 32 & 33 \\
\end{array}
\]

\[
\begin{array}{cccc}
0 & 1 & 33 & 199 \\
\end{array}
\]
More pointer arithmetic

- address one past the end of an array is ok for *pointer comparison only*

- what about *(begin+44)?

- what does begin++ mean?

- how are pointers compared using < and using ==?

- what is value of end - begin?

```
char * a = new int[44];
char * begin = a;
char * end = a + 44;

while (begin < end)
{
    *begin = 'z';
    begin++;
    // *begin++ = 'z'
}
```
What is a C-style string?

- array of char terminated by sentinel ‘\0’ char
  - sentinel char facilitates string functions
  - ‘\0’ is nul char, unfortunate terminology
  - how big an array is needed for string “hello”?

- a string is a pointer to the first character just as an array is a pointer to the first element
  - char * s = new char[6];
  - what is the value of s? of s[0]?

- char * string functions in <string.h>
C style strings/string functions

- **strlen** is the # of characters in a string
  - same as # elements in char array?

  ```c
  int strlen(char * s)
  // pre: \0 terminated
  // post: returns # chars
  {
    int count=0;
    while (*s++ ) count++;
    return count;
  }
  ```

- Are these less cryptic?

  ```c
  while (s[count]) count++;
  // OR, is this right?
  char * t = s;
  while (*t++);
  return t-s;
  ```

- what’s “wrong” with this code?

  ```c
  int countQs(char * s)
  // pre: \0 terminated
  // post: returns # q’s
  {
    int count=0;
    for(k=0;k < strlen(s);k++)
      if (s[k]==’q’) count++;
    return count;
  }
  ```

- how many chars examined for 10 character string?

- solution?
More string functions (from < string.h>)

- **strcpy copies strings**
  - who supplies storage?
  - what’s wrong with \texttt{s = t}?

```c
char s[5];
char t[6];
char * h = “hello”;
strcpy(s,h); // trouble!
strcpy(t,h); // ok
```

- **strncpy copies n chars (safer?)**

- **what about relational operators <, ==, etc.?**
- can’t overload operators for pointers, no overloaded operators in C
- **strcmp (also strncmp)**
  - return 0 if equal
  - return neg if lhs < rhs
  - return pos if lhs > rhs

```c
if (strcmp(s,t)==0) // equal
if (strcmp(s,t) < 0)// less
if (strcmp(s,t) > 0)// ????
```
Arrays and pointers

- These definitions are related, but not the same
  ```cpp
  int a[100];
  int * ap = new int[10];
  ```
- Both `a` and `ap` represent ‘arrays’, but `ap` is an lvalue

- Arrays converted to pointers for function calls:
  ```cpp
  char s[] = "hello";
  // prototype: int strlen(char * sp);
  cout << strlen(s) << endl;
  ```

- Multidimensional arrays and arrays of arrays
  ```cpp
  int a[20][5];
  int * b[10]; for(k=0; k < 10; k++) b[k] = new int[30];
  ```
Microsoft question

- Write atoi, write itoa, which is harder?

- Questions? Issues? Problems?

```c
int atoi(const char * sp);
char * itoa(int num);
```

- **Difference between** `const char * p` and `char * const p`
  - one is a pointer to a constant character
  - one is a constant pointer to a character