Lecture 10

- Pointers continued
- C-strings
  - Representation
  - Library functions

Pointers and Structures

- Two equivalent ways to access structure members via pointers:
  - (*p).member
  - p->member

```
struct Point { int x,y; } point, *pp;

pp->x = point.x;
pp->y = point.y;

(*pp).x = point.x;
(*pp).y = point.y;  // equivalent

*pp = point;        // equivalent
```


Example: Trees

- Trees are a special kind of graph
- Graphs consist of nodes and edges that connect two nodes
- Trees: all nodes are connected, no cycles
- In computing science, trees are fundamental dynamic data structures:
  - Data is associated with nodes
  - Nodes contain pointers to successor nodes
- Example: Binary Trees
  (nodes have at most two successors)


```
// binary tree: nodes have at most two successors

struct Node {
   int data;           // data associated with node
   Node *left, *right; // pointers to successor nodes
};                     // 0 indicates no successor

// create small tree:     root
//                        /  \
//                       a    b

Node *root = new Node; // all components undefined!
Node *a    = new Node;
Node *b    = new Node;

// *a and *b have no successors (they are "leaves")
a->left = a->right = b->left = b->right = 0;

// connect sucessor nodes a and b to root
root->left = a;  root->right = b;
```

Deleting Trees

```
// deleting trees recursively in reverse order
// "what is connected last gets deleted first"

// precondition: n points to the root of a tree

void delete_tree(Node *n)
{
  if (n == 0) return;    // nothing to delete
  delete_tree(n->left);  // delete left subtree
  delete_tree(n->right); // delete right subtree
  delete n;              // finally, delete node
}
```


Pointer Arrays, Pointers to Pointers

```
int *a[4]; // array of 4 pointers to int

float **b; // b is a pointer to a pointer to a float

a[0] -> int        b -> b' -> float
a[1] -> int
a[2] -> int
a[3] -> int
```

- Pointers are variables themselves, thus
  - they can be stored in arrays, and
  - can point to pointers

C-Strings: Constants

- A C-string is a sequence of characters
- C-string constants are double-quoted
  - cout << "I am a string" << endl;
  - cout << "hello world\n";
  - can contain escape sequences such as **\n** or **\a**
  - " in the text is represented by \", e.g. cout << "\"";

C-String Representation

```
char s[9] = "Hello!";

s[0] s[1] s[2] s[3] s[4] s[5] s[6] s[7] s[8]
  H    e    l    l    o    !   \0  <both undef>

char s[] = "Hello!";// reserves enough space to
                    // hold Hello! + \0
-> sizeof(s) = 7
```

- Array of characters which contains the character sequence
- Plus **end-marker** '\0' (0 byte)
- **Inefficient**! (why?) C++ comes with a more sophisticated string template class (later)
- C-strings can be initialized via =

C-String Pitfalls

- Ensure that the char array is **big enough** - must hold characters + end-marker 0!
- Character with code 0 cannot be represented in a C-string because 0 indicates end-of-string
- Assignments other than initializations are illegal
- == and other relational operators don't work with C-strings
- **Does not sound very useful**
- **Solution: library functions!**

C-String Library Functions (<cstring>)

- **int strlen(const char s[]);**
  - returns the # of characters in s excluding the end-marker
- **void strcpy(char dest[], const char src[]);**
  - copies string src to dest **(dest must be large enough!)**
- **int strcmp(const char s1[], const char s2[]);**
  - compares strings s1 and s2
  - returns 0 iff they are equal
  - return number > 0 iff s1 > s2 (lexicographical order)
  - return number < 0 iff s1 < s2
- **void strcat(char dest[], const char src[]);**
  - appends string src to dest overwriting its end-marker and adds '\0'

String Assignment, Comparison

```
#include <cstring>
char s1[] = “hello”;
char s2[100];

strcpy(s2, s1);  // s2 equals “hello”

char s_too_short[2];

strcpy(s_too_short, s1); // undefined!

char a[] = “aaa”;
char b[] = “aaaa”;
char c[] = “b”;

strcmp(a, a) == 0
strcmp(a, b) < 0
strcmp(c, b) > 0

strlen(b) == 4
```


strlen & strcpy Implementaton

```
// returns number of characters, array version
int strlen(const char s[])
{
  int i=0;
  while (s[i]) ++i;
  return i;
}

// copies src to dest, array version
void strcpy(char dest[], const char src[])
{
  int i=0;
  char c;
  do {
    c = dest[i] = src[i];
    ++i;
  } while (c);
}
```

```
// return length of string, pointer version
int strlen(const char *s)
{
  const char *p = s;
  while (*p) ++p;
  return p-s; // pointer arithmetic
}


// copy t to s, pointer version
void strcpy(char *s, const char *t)
{
  while (*s++ = *t++);
}
```



```
// return 0 if strings are equal
// < 0 if s1 < s2 (lexicographically)
// > 0 if s1 > s2 (lexicographically)
int strcmp(const char s1[], const char s2[])
{
  int i = 0;
  unsigned char c1, c2;

  // scan s1 and s2 until
  // either a end of string is reached
  // or characters are different

  do {
    c1 = s1[i]; c2 = s2[i]; ++i;
    if (c1 == 0) break;
  } while (c1 == c2);

  return c1-c2;
}
```


strcat Implementaton

```
// appends the src string to the dest string
// overwriting the '\0' character at the end of
// dest and then adds a terminating '\0' character

void strcat(char dest[], const char src[])
{
  int i=0;
  while (dest[i]) ++i;  // find end-marker
  int j=0;
  char c;
  do {
    c = dest[i++] = src[j++]; // append src
  } while (c);
}
```