



DEPARTMENT OF INFORMATION SYSTEMS AND COMPUTER SCIENCE



```
001011100100011110111100100110101110100100101  
11010101101010100001010101010010101010101010  
10100101001001001010101010101010101010101010  
11100001111010110000000111101010101010000010101  
111010101110010100010010111010100010100100111010  
10101001010010010010000101010110101010101010010111  
00101010010101001010100000001010101001111101000011001  
1000110010000111100110101011000100110101010000101010  
11001010101010000100110010101000100101010101010  
101001010010010010101010101010101010101010101010  
1110000111101011000000011110101010101010000010101  
00100101010010100100101001000101010101010101010010  
1001010010000101010010010101001010010101010010010  
1001010010101010101001010101010101010101001001001  
10010101010101010101010101010101010101010101010
```

										01
										02
										03
										04
										05
A	B	C	D	E	F	G	H			

Lists

CS179.14A Survival Guide II

Lecture Time!

- ▶ Review: Dynamic Allocation
- ▶ Lists
- ▶ Arrays of Lists of Pointers

0010101001010100001111010000100
10001100100001111001101010010101
11001010101010100001001100101010100
100101001001001010101010101010101
11100001111010110000000111101001
001001010100101001001010010010110
10010100100001010100100101001010
1001010010101001010010101010101



DISCS

Review: Dynamic Allocation

- ▶ When you need to defer initialization for any reason, you will have to rely on *dynamic allocation*
 - ▶ array size known only during run-time
 - ▶ nodes in linked lists
 - ▶ etc.

00101010010101000011110100001100
10001100100001111001101010010101
11001010101010100001001100101010100
100101001001001010101010101010101
11100001111010110000000111101001
001001010100101001001010010010110
1001010010001010100100101001010
100101001010100101001010010101



Review: Dynamic Allocation

- ▶ However, these variables are explicitly allocated through the `new` keyword
- ▶ And they must also be explicitly deallocated through the `delete` keyword
- ▶ Failure to properly deallocate will result in memory leaks
- ▶ The O/S usually deallocates them once the program terminates

001010100101010001111001101010010101
110010101010100001001100101010100
100101001001001010101010101010101
11100001111010110000000111101001
001001010100101001001010010010110
1001010010001010100100101001010
100101001010100101001010010101



Dynamic Allocation

```
int* p = new int;  
*p = 28;  
cout << *p << "\n";  
delete p;
```

```
int size = 5;  
int* arr = new int[size];  
arr[1] = 12345;  
cout << arr[1] << "\n";  
delete[] arr;
```

0010101001010100011110100001100
10001100100001111001101010010101
110010101010100001001100101010100
1001010010010010101010101010101
11100001111010110000000111101001
001001010100101001001010010010110
10010100100001010100100101001010
10010100101010010100101010010101



DISCS

Lists

- ▶ A *list* is a sequence container implemented as a doubly-linked list
- ▶ You can also use another container type (`vector`, etc.) that can act as a resizable array, but `list` generally performs better in inserting, extracting, and moving elements in any position within it

0010101001010100011110100001100
10001100100001111001101010010101
110010101010100001001100101010100
100101001001001010101010101010101
11100001111010110000000111101001
001001010100101001001010010010110
10010100100001010100100101001010
10010100101010010100101001010101



DISCS

Lists

```
#include <list>
```

```
list<MyEntity*> myList;    // list of pointers
```

```
// in some function
```

```
MyEntity me;
```

```
MyEntity* p = &me;
```

```
// ...
```

```
myList.push_back( p );    // add p at end of list
```

```
myList.remove( p );       // remove all equal to p
```

Lists



```
for (
```

```
list<MyEntity*>::iterator it = myList.begin();
it != myList.end(); ++it )
```

{

```
cout << (**it).hp << "out of ";
```

```
cout << (**it).maxhp << " HP left.\n";
```

}

Array of Lists

- ▶ But what if you want an array of lists?
 - ▶ one list for each node in a tree
 - ▶ one list for each cell in a grid
 - ▶ ... etc.
- ▶ And what if that array's size is to be determined at run-time?

0010101001010100011110100001100
10001100100001111001101010010101
11001010101010100001001100101010100
100101001001001010101010101010101
11100001111010110000000111101001
001001010100101001001010010010110
10010100100001010100100101001010
10010100101010010100101001010101



DISCS

Array of Lists

```
list<MyEntity*>* row;
```

```
// in some function
```

```
row = new list<MyEntity*>[SIZE];
```

```
// for 2-D arrays, create an array of arrays
```

```
// of lists, then initialize each of these
```

```
// sub-arrays
```

```
// (there's a reason I named the variable "row")
```



Array of Lists

```
// upon program termination
```

```
delete[] row;
```

```
// for 2-D arrays, you must delete each of the
```

```
// sub-arrays first before deleting the main
```

```
// array
```

```
0010101001010100001111001101010010101  
10001100100001111001101010010101  
11001010101010100001001100101010100  
100101001001001010101010101010101  
11100001111010110000000111101001  
001001010100101001001010010010010110  
10010100100001010100100101001010  
10010100101010010100101001010101  
10010100101010010100101001010101
```



DISCS