• Exam 1 on July 18, 2005 – 10:00-11:40am

Pointers

Addresses in Memory

• When a variable is declared, enough memory to hold a value of that type is allocated for it at an unused memory location. This is the address of the variable.

```c
int x; float number; char ch;
```

```plaintext
+-----+-------+-----+
| 2000| 2004  | 2012|
| x   | number| ch  |
+-----+-------+-----+
```
Obtaining Memory Addresses

- The address of a variable can be obtained by using the address-of operator ‘&’

```cpp
int x;
float number;
char ch;
cout << "Address of x is " << &x << endl;
cout << "Address of number is " << &number << endl;
cout << "Address of ch is " << &ch << endl;
```

What is a Pointer Variable?

- A pointer variable is a variable whose value is the address of a location in memory
- To declare a pointer variable, you must specify the type of value that the pointer will point to.

```cpp
int *ptr; // ptr will hold the address of an int
char *q;  // q will hold the address of a char
```

Using a Pointer Variable

- Because ptr holds the address of x, we say that ptr “points to” x

```cpp
int x;
x = 12;
int *ptr;
ptr = &x;
```
Unary operator * is the deference operator

- The value pointed to by ptr is denoted by *ptr

Using the dereference operator

- *ptr=5 changes the value at address ptr to 5

Another Example

- char ch;
  ch = 'A';
  char *q;
  q = &ch;
  "q = 'Z';
  char *p;
  p = q
The NULL Pointer

- There is a pointer constant 0 called the "null pointer" denoted by NULL in cstddef.
- But NULL is not memory address 0.
- It is an error to dereference a pointer whose value is NULL. Such an error may cause your program to crash, or behave erratically. It is the programmer's job to check for this.

```cpp
if (ptr != NULL)
    {
        ... // ok to use *ptr here
    }
```

Allocation of Memory

<table>
<thead>
<tr>
<th>STATIC ALLOCATION</th>
<th>DYNAMIC ALLOCATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static allocation is the allocation of memory space at compile time.</td>
<td>Dynamic allocation is the allocation of memory space at run time by using operator new.</td>
</tr>
</tbody>
</table>
Three Kinds of Program Data

• **STATIC DATA**: memory allocation exists throughout execution of program.
  - static long SeedValue;

• **AUTOMATIC DATA**: automatically created at function entry, resides in activation frame of the function, and is destroyed when returning from function.

• **DYNAMIC DATA**: explicitly allocated and deallocated during program execution by C++ instructions written by programmer using unary operators `new` and `delete`.

Using Operator `new`

• If memory is available in an area called the free store (or heap), operator `new` allocates the requested object or array, and returns a pointer to (address of) the memory allocated.

• Otherwise, the null pointer 0 is returned.

• The dynamically allocated object exists until the `delete` operator destroys it.

Dynamically Allocated Data

```cpp
char *ptr;
ptr = new char;
*ptr = 'B';
cout << *ptr;
```
Dynamically Allocated Data

```
char* ptr;
ptr = new char;
*ptr = 'B';
cout << *ptr;
```

```
char* ptr;
ptr = new char;
*ptr = 'B';
cout << *ptr;
```

```
char* ptr;
ptr = new char;
*ptr = 'B';
cout << *ptr;
delete ptr;
```

Delete deallocates the memory pointed to by ptr
- The pointer is considered unassigned. The memory is returned to the free store.
Dynamic Arrays

- Array limitations
  - Must specify size first
  - May not know until program runs!
- Must ‘estimate’ maximum size needed
  - Sometimes OK, sometimes not
  - ‘Wastes’ memory
- Dynamic arrays
  - Can grow and shrink as needed

Dynamic Array Allocation

```c
char *ptr; // ptr is a pointer variable that can hold
// the address of a char

ptr = new char[5];
// dynamically, during run time, allocates memory for 5
// characters and places into the contents of ptr their
// beginning address
```

![Dynamic Array Allocation Diagram 1](image1)

```c
ptr = new char[5];
strncpy(ptr, "Bye");
```

![Dynamic Array Allocation Diagram 2](image2)
Deleting Dynamic Arrays

- Allocated dynamically at run-time
  - So should be destroyed at run-time
- Use "delete [] ptr;"
  - De-allocates all memory for dynamic array
  - Brackets indicate 'array' is there

Memory Leak

- A memory leak occurs when dynamic memory (that was created using operator new) has been left without a pointer to it by the programmer, and so is inaccessible.

Causing a Memory Leak

```cpp
int *ptr = new int;
*ptr = 8;
int *ptr2 = new int;
*ptr2 = -5;
ptr = ptr2;
```
A Dangling Pointer

- Occurs when multiple pointers point to the same object and delete is applied to one of them.

Leaving a Dangling Pointer

```
int *ptr = new int;
*ptr = 8;
int *ptr2 = new int;
*ptr2 = -5;
ptr = ptr2;
ptr
ptr2

delete ptr2;
ptr2 = NULL;
```

Dynamic Array Example

Sort an array of float numbers from standard input. The first element is the size of the array. For example:

```
5 3.4 2.6 5.7 100.4 43.5
```

```
sortFromInput() {
    float *array;
    int size, idx;
    cin >> size;
    array = new float[size];
    for (idx=0; idx<size; idx++)
        cin >> array[idx];
    Sort(array, size);
    OutputSortedArray(array, size);
    delete[] array;
}
```