1 Dynamic Memory

The use of dynamic memory is a very powerful and common programming technique. Many books have an entire chapter devoted to the topic of memory (management, manipulation, etc.).

All *modern* programming languages support dynamic memory management in some way:

- Explicit programmer control (C/C++)
- Environment control (Java, C#, Scheme)
- A combination (Objective-C)

Manipulating memory dynamically **requires** considerable attention by the programmer.

2 Pointers

2.1 Basic Pointer Concepts

What are pointers?

A *pointer variable*, or simply a *pointer* can reference a memory cell. The reference to another memory cell is the computer's representation of the location, or address in memory, of the cell.

How are they used?

- Point to a memory location.
- Call by reference is based on pointers.
- Operators:
  - `&` *Address of* operator
  - `*` Dereferencing (*contents of*) operator
- Machine/compiler dependencies exist.
- Care and caution should be exercised when using pointers!

Pointers will be used extensively in later Computer Science courses—unless everything moves to Java.

2.2 Pointer Examples

```
int  a;
int  *aPtr;

a = 5;
cout << a << endl;
aPtr = &a;
cout << *aPtr << endl;    //  contents of a
*aPtr = 6;
cout << a << endl;
cout << *aPtr << endl;    //  contents of a
cout << &a    << endl;    //  address of a
                          //    (compiler/machine dependent)
```

Output:

```
5
5
6
6
0x024b2fa8
```

This example is in my old *C++ Notes*, page 137.

2.3 Arrays and Pointers

```
int  a[5] = { 5, 10, 15, 20, 25 };
int  *aPtr;

aPtr = a;
cout << *aPtr << endl;
aPtr = &a[0];
cout << *aPtr << endl;
aPtr = &a[2];
cout << *aPtr << endl;
```

Output:

```
5
5
15
```

2.4 More Arrays and Pointers

Pointer arithmetic.

```
int  a[5] = { 1, 3, 5, 7, 11 };
int  *aPtr;

aPtr = a;
aPtr += 3;      //  advance aPtr by 3
cout << *aPtr << endl;
cout << a[3]  << endl;
```

Output:

```
7
7
```

2.5 Motivation for using Pointers

Arrays:

- Fixed size (`N` at compile time)
- Homogeneous (same type)
- Access items using an index (range `0..N-1`)
- Stored in contiguous memory locations

Linked Lists:

- Dynamic (change at run time)
- Homogeneous (typically)
- Access items using pointers
- Not necessarily stored in contiguous memory locations

2.6 Declaring Pointer Variables

The declaration

```
int*  p;     // or
int  *p;
```

declares `p` to be an integer pointer variable; that is, `p` can point only to memory cells that contain integers. Pointers can be declared to *any* type except files. [Note: The C programming language uses the type `FILE *` as a file pointer. See K&R for more details.]

Care must be used when declaring pointer variables!

In reality,

Care must be used when using pointers!

The declaration

```
int*  p, q;
```

declares p to be a pointer to an integer, but declares q to be an integer.

The correct way to declare both p and q as integer pointer variables is

```
int  *p, *q;
               // or
int* p;
int* q;
```

The latter makes commenting easier and it is obvious that both p and q are pointers to integers.

2.7 Memory Allocation: Static

Two pointer variables

```
int* p;
int* q;
```

the memory for the pointer variables above is allocated at compile time, that is, before the program executes. This type of memory allocation is called *static allocation* and the variables are called *statically allocated variables.*

2.8 Memory Allocation: Dynamic

Memory allocation can also occur at run time (execution) and is called *dynamic allocation.* A variable that is allocated then is called a *dynamically allocated variable.*

2.9 Dynamic Memory Allocation in C++

C++ enables dynamic memory using the operator `new`, which acts on a data type,

```
int *p = new int;
```

`new int` allocates a memory cell that can contain an integer and returns a pointer to the new cell. **The initial content of this new cell is undetermined**.

The newly created (allocated) memory cell has no programmer defined name. The only way to access its content or to put a value in it is indirectly via the pointer that `new` creates.

2.10 Releasing Dynamic Memory in C++

C++ provides the operator `delete` to release memory.

```
delete P;
```

2.11 Arrays (Static) in C++

```
const int A_SIZE = 50;
int A[A_SIZE];
```

An array name is a pointer to the first element of the array

`*A` is equivalent to `A[0]`
`*(A+1)` is equivalent to `A[1]`
....

2.12 Dynamic Arrays in C++

```
int A_SIZE = 50;                  // not a constant
int *pIArr = new int[A_SIZE];
```

`pIArr` can be treated as an array, e.g., `pIArr[i]`, etc.

Release the memory allocated for the array when finished using it.

```
delete [] pIArr;
```

Data Cleaning

Pull out just the names from a data file that is defined as follows:

```
1 0 Adams, Mary F. 123-45534 **Web Registered** Undergraduate
3.000 Enter Enter   E-mail
2 0 Badenuff, Boris M. 172-52637 **Web Registered** Undergraduate
3.000 Enter Enter   E-mail
3 0 Karson, John A. 051-55391 **Web Registered** Non-Degree
3.000 Enter Enter   E-mail
4 0 Zebra, Red T. 155-34748 **Web Registered** Undergraduate
3.000 Enter Enter No E-mail
```