C++ Dynamic Memory

Memory can be allocated dynamically (i.e., during program execution). The requested memory does NOT have a name (like a regular variable), but instead it is referenced via a pointer.

```c
int *p;
p = new int;
*p = 3;
```

The `new` operator returns a pointer to the dynamic memory

---

C++ Dynamic Arrays

```c
int *p;
p = new int[100];
for(int i=0; i<100; i++)
p[i] = 5;
```

NOTE: `p` is really a pointer (to int), not an array. However, the “subscript” form of pointer arithmetic can be used, as seen in previous examples.

Therefore, it appears that "arrays" can be dynamically allocated!
Freeing Dynamic Memory

After a program is done with dynamic memory, it should be "given back" to the system:

```cpp
delete p;
```

This releases the memory pointed to by `p` back to the system.

C++ Dynamic structs

```cpp
struct emtype
{
    char name[20];
    int id;
    float salary;
};

emtype *pemp;

pemp = new emtype;
```

[Diagram showing memory allocation and pointer to `emtype` structure]
C++ Dynamic structs

To access struct members:

```c
(*pemp).id = 12345;
strcpy((*pemp).name, "Joe Blow");
(*pemp).salary = 1.98;
```

A new notation:

```c
pemp->id = 12345;
strcpy(pemp->name, "Joe Blow");
pemp->salary = 1.98;
```

The operator hierarchy of C makes the {}'s necessary in the first cases above. The alternate -> notation can be used instead. The two notations are equivalent.

---

Operator Hierarchy Chart (Complete)

<table>
<thead>
<tr>
<th>Operators</th>
<th>Notes</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>( [ ] )</td>
<td></td>
<td>Left to Right</td>
</tr>
<tr>
<td>1 ~ ++ -- + - * &amp; (type) sizeof</td>
<td>Unary ops</td>
<td>Right to Left</td>
</tr>
<tr>
<td>/ *</td>
<td>Arith ops</td>
<td>Left to Right</td>
</tr>
<tr>
<td>+ -</td>
<td>Arith ops</td>
<td>Left to Right</td>
</tr>
<tr>
<td>&lt;&lt; &gt;&gt;</td>
<td>Binary shifts</td>
<td>Left to Right</td>
</tr>
<tr>
<td>&lt; &lt;= &gt; &gt;=</td>
<td></td>
<td>Left to Right</td>
</tr>
<tr>
<td>== !=</td>
<td></td>
<td>Left to Right</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>Logical AND</td>
<td>Left to Right</td>
</tr>
<tr>
<td></td>
<td>Logical EX-OR</td>
<td>Left to Right</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>? :</td>
<td>Decision OP</td>
<td>Right to Left</td>
</tr>
<tr>
<td>= += -= /= %= &amp;= ^=</td>
<td>= &lt;&lt;= &gt;&gt;</td>
<td>Comma</td>
</tr>
</tbody>
</table>