11.9: Pointers to Structures

- **Given the following Structure:**

  ```
  struct Student {
    string name;      // Student’s name
    int idNum;        // Student ID number
    int creditHours;  // Credit hours enrolled
    float gpa;        // Current GPA
  }
  ```

- **We can define a pointer to a structure**

  ```
  Student s1 = {"Jane Doe", 12345, 15, 3.3};
  Student *studentPtr;
  studentPtr = &s1;
  ```

- **Now studentPtr points to the s1 structure.**

---

**Pointers to Structures**

- **How to access a member through the pointer?**

  ```
  Student s1 = {"Jane Doe", 12345, 15, 3.3};
  Student *studentPtr;
  studentPtr = &s1;
  cout << *studentPtr.name << end;        // ERROR
  ```

  - *studentPtr.name* is equivalent to *(studentPtr.name)
  - So this will work:

  ```
  cout << (*studentPtr).name << end;     // WORKS
  ```

- **structure pointer operator: -- >**

  - Due to the “awkwardness” of the notation, C has provided an operator for dereferencing structure pointers:

    ```
    studentPtr->name is equivalent to (*studentPtr).name
    ```

  - The structure pointer operator is the hyphen (-) followed by the greater than (>): like an arrow.

  - In summary:

    ```
    s1.name     // member of structure s1
    sptr->name  // member of a structure pointed to by sptr
    ```
Structure Pointer: example

• Function to input a student, using a ptr to struct

```c
void getStudent(Student *s) {
    cout << "Enter Student name: ";
    getline(cin,s->name);
    cout << "Enter studentID: ";
    cin >> s->idNum;
    cout << "Enter credit hours: ";
    cin >> s->creditHours;
    cout << "Enter GPA: ";
    cin >> s->gpa;
}
```

• Call:

```c
Student s1;
getStudent(&s1);
```

Dynamically Allocating Structures

• Structures can be dynamically allocated with new:

```c
Student *sptr;
sptr = new Student;
sptr->name = "Jane Doe";
sptr->idNum = 12345;
...
delete sptr;
```

• Arrays of structures can also be dynamically allocated:

```c
Student *sptr[100];
sptr[0].name = "John Deer";
...
delete [] sptr;
```

Structures and Pointers

• Expressions:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s-&gt;m</td>
<td>s is a structure pointer, m is a member</td>
</tr>
<tr>
<td>*a.p</td>
<td>a is a structure, p (a pointer) is a member. This expr is the value pointed to by p: *(a.p)</td>
</tr>
<tr>
<td>(*s).m</td>
<td>s is a structure pointer, m is a member. Equivalent to s-&gt;m</td>
</tr>
<tr>
<td>*s-&gt;p</td>
<td>s is a structure pointer, and p (a pointer) is in the structure pointed to by s. Equiv to *(s-&gt;p).</td>
</tr>
<tr>
<td>(*(*s).p)</td>
<td>s is a structure pointer, and p (a pointer) is in the structure pointed to by s. Equiv to *(s-&gt;p).</td>
</tr>
</tbody>
</table>

in 13.3: Pointers to Objects

• We can define pointers to objects, just like pointers to structures

```c
Time t1(12,20);
Time *timePtr;
timePtr = &t1;
timePtr->addMinute();
cout << timePtr->display() << endl;
```

• We can access public members of the object using the structure pointer operator (->)

```c
timePtr->addMinute();
cout << timePtr->display() << endl;
```

Output: 12:21
Dynamically Allocating Objects

- Objects can be dynamically allocated with new:

```cpp
Time *tptr;
tptr = new Time(12,20);
...
delete tptr;
```

- Arrays of objects can also be dynamically allocated:

```cpp
Time *tptr;
tptr = new Time[100];
tptr[0].addMinute();
...
delete [] tptr;
```

You can pass arguments to a constructor using this syntax.

It can use only the default constructor to initialize the elements in the new array.

Deleting Dynamically Allocated Objects

- Recall that whenever an object is “destroyed” that its destructor is called.
  - Automatic/regular variables are destroyed at the end of their scope (end of block/function where they are defined).
  - Dynamically allocated objects are destroyed when they are “deleted”.
- If an object contains dynamically allocated variables that are deleted in its destructor (like in IntCell), then they will be deleted when the containing object is deleted.

Deleting Dynamically Allocated Objects

- Recall IntCell, with dynamically allocated value.

```cpp
class IntCell
{
    private:
        int *storedValue;
    public:
        IntCell(int);
        ~IntCell();
        int read();
        void write(int);
};
IntCell::IntCell(int val) {
    storedValue = new int;
    *storedValue = val;
}
IntCell::~IntCell() {
    delete storedValue;
}
```

Deleting Dynamically Allocated Objects

- driver that has a ptr to IntCell:

```cpp
#include "IntCell.h"
int main() {
    IntCell *icptr;
icptr = new IntCell(5);
cout << icptr->read() << endl;
delete icptr;
//...
return 0;
}
```

This calls the destructor first, which deletes (deallocates) icptr->storedValue.
13
in 14.5 The this pointer

- **this**: a predefined pointer available to a class’s member functions
- **this** always points to the instance (object) of the class whose function is being called.
- When used inside a member function of the Time class (for example), it has this invisible declaration:

  ```cpp
  Time *this;
  ```

14
this: access hidden members

- You can use **this** to access members that may be hidden by parameters with the same name (especially in constructors/setters):

```cpp
Time::Time(int hour, int minute) {
    this->hour = hour;
    this->minute = minute;
}
```

15
this: an object can return itself

- Often, an object will return itself as the result of a binary operation, like assignment:

  ```cpp
  v1 = v2 = x;  // is equivalent to  v1 = (v2 = x);
  ```

- because associativity of = is right to left.
- But what is the result of (v2 = x)?

- It is the left-hand operand, v2.

  ```cpp
  v1 = v2 = x;  // is equivalent to  v2 = x;
  v1 = v2;
  ```

16
Returning this

```cpp
class Time {
    private:
    int hour, minute;
    public:
    const Time operator= (const Time &right) {
        hour = right.hour;
        minute = right.minute;
        return *this;
    }
};

const Time Time::operator= (const Time &right) {
    hour = right.hour;
    minute = right.minute;
    return *this;
}
```

Time time1, time2, time3(2,25);
Time time1 = time2 = time3;
```cpp
```