Even More C++
Outline

- Dynamic Memory
- Data Structures
- Other Data Types
A data structure is a group of data elements grouped together under one name

- Not quite the same thing as a data type
- Use struct to define a structure in C++

```c++
struct type_name {
    member_type1 member_name1;
    member_type2 member_name2;
    member_type3 member_name3;
    
} object_names;

struct product {
    int weight;
    double price;
} ;

product apple;
product banana, melon;

apple.weight
apple.price
banana.weight
banana.price
melon.weight
melon.price
```

Accessing data elements in a structure
An Example of struct

```cpp
// example about structures
#include <iostream>
#include <string>
#include <sstream>
using namespace std;

struct movies_t {
    string title;
    int year;
} mine, yours;

void printmovie (movies_t movie);

int main ()
{
    string mystr;
    mine.title = "2001 A Space Odyssey";
    mine.year = 1968;
    cout << "Enter title: ";
    getline (cin,yours.title);
    cout << "Enter year: ";
    getline (cin,mystr);
    stringstream(mystr) >> yours.year;
    cout << "My favorite movie is:\n ";
    printmovie (mine);
    cout << "And yours is:\n ";
    printmovie (yours);
    return 0;
}

void printmovie (movies_t movie)
{
    cout << movie.title;
    cout << " (" << movie.year << ")\n";
}
An Example of an Array of structs

```cpp
// array of structures
#include <iostream>
#include <string>
#include <sstream>
using namespace std;

struct movies_t {
    string title;
    int year;
} films [3];

void printmovie (movies_t movie);

int main ()
{
    string mysttr;
    int n;

    for (n=0; n<3; n++)
    {
        cout << "Enter title: ";
        getline (cin,films[n].title);
        cout << "Enter year: ";
        getline (cin,mysttr);
        stringstream(mysttr) >> films[n].year;
    }

    cout << "You have entered these movies:
    for (n=0; n<3; n++)
        printmovie (films[n]);
    return 0;
}

void printmovie (movies_t movie)
{
    cout << movie.title;
    cout << " (" << movie.year << ")\n";
}
```

Enter title: Blade Runner
Enter year: 1982
Enter title: The Matrix
Enter year: 1999
Enter title: Taxi Driver
Enter year: 1976

You have entered these movies:
Blade Runner (1982)
The Matrix (1999)
Taxi Driver (1976)
Pointers to Structures

- The arrow operator `->` is used to access structures that have member elements.

```c
struct movies_t {
    string title;
    int year;
};

movies_t amovie;
movies_t * pmovie;

pmovie = &amovie;

pmovie->title

*pmovie.title
```

is equivalent to:

```c
(*pmovie).title

(*pmovie.title)
```

<table>
<thead>
<tr>
<th>Expression</th>
<th>What is evaluated</th>
<th>Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.b</td>
<td>Member b of object a</td>
<td></td>
</tr>
<tr>
<td>a-&gt;b</td>
<td>Member b of object pointed to by a</td>
<td>(*a).b</td>
</tr>
<tr>
<td>*a.b</td>
<td>Value pointed to by member b of object a</td>
<td>*(a.b)</td>
</tr>
</tbody>
</table>
Nesting Structures

- Structures can be nested within other structures

```c
struct movies_t {
    string title;
    int year;
};

struct friends_t {
    string name;
    string email;
    movies_t favorite_movie;
} charlie, maria;

friends_t * pfriends = &charlie;

charlie.name
maria.favorite_movie.title
charlie.favorite_movie.year
pfriends->favorite_movie.year
```
Other Data Types: Aliases

- Two ways to create a type alias:
  - `typedef existing_type new_type_name;`
  - `using new_type_name = existing_type;`

- “using” is more generic, but “typedef” is likely found more often in existing code
Other Data Types: Unions

- Declaration similar to struct, but meaning is very different

```c
union mytypes_t {
    char c;
    int i;
    float f;
} mytypes;
```

- All three member elements use the same memory space
  - Memory allocated is the size of the largest
    - In the example above, probably the size of a float

- Use this when you want to access an element in its entirety or as an array of smaller elements
Union Example

```c
union mix_t {
  int l;
  struct {
    short hi;
    short lo;
  } s;
  char c[4];
} mix;
```
## Anonymous Unions

<table>
<thead>
<tr>
<th>structure with regular union</th>
<th>structure with anonymous union</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct book1_t {</td>
<td>struct book2_t {</td>
</tr>
<tr>
<td>char title[50];</td>
<td>char title[50];</td>
</tr>
<tr>
<td>char author[50];</td>
<td>char author[50];</td>
</tr>
<tr>
<td>union {</td>
<td>union {</td>
</tr>
<tr>
<td>float dollars;</td>
<td>float dollars;</td>
</tr>
<tr>
<td>int yen;</td>
<td>int yen;</td>
</tr>
<tr>
<td>} price;</td>
<td>}</td>
</tr>
<tr>
<td>} book1;</td>
<td>} book2;</td>
</tr>
</tbody>
</table>

- `book1.price.dollars`
- `book1.price.yen`
- `book2.dollars`
- `book2.yen`
Enumerated Types (enum)

```c
enum type_name {
    value1,
    value2,
    value3,
    ...
} object_names;

enum colors_t {black, blue, green, cyan, red, purple, yellow, white};

colors_t mycolor;
mycolor = blue;
if (mycolor == green) mycolor = red;
```

- You can use these names or their integer equivalents
Enumerated Types (enum class)

```cpp
enum class Colors {black, blue, green, cyan, red, purple, yellow, white};

Colors mycolor;
mycolor = Colors::blue;
if (mycolor == Colors::green) mycolor = Colors::red;
```
Summary

- Dynamic Memory
- Data Structures
- Other Data Types