Structures

• Structured Data types
• Data abstraction
• struct's
• Array of structs
Group together multiple simple items in a single object
Why Structured Types?

- Group item together

- Entries in the campus directory
  - Last Name
  - First Name
  - Phone Number
  - Address
  - Major
Why Structured Types?

- Group items together

- Cards:
  - Rank
    - A, K, Q, J, 10, 9, 8, 7, 6, 5, 4, 3, 2
  - Suit
    - Clubs, Diamonds, Hearts, Spades
Why Structured Types?

- Group items together
- FootBall Game
  - season
  - opponent
  - location
  - win/lose/tie
  - UM score
  - opponent score
Picturing a single game

Season: 2006
Opponent: "Vanderbilt"
Location: Ann Arbor, MI
Win/Lose/Tie: W
UM Score: 27
Opponent Score: 7
<table>
<thead>
<tr>
<th>Season</th>
<th>2003</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opponent</td>
<td>&quot;Ohio State University&quot;</td>
</tr>
<tr>
<td>Location</td>
<td>Ann Arbor, MI</td>
</tr>
<tr>
<td>Win/Lose/Tie</td>
<td>W</td>
</tr>
<tr>
<td>UM Score</td>
<td>35</td>
</tr>
<tr>
<td>Opponent Score</td>
<td>21</td>
</tr>
</tbody>
</table>
struct (ADT record)

A struct groups related data together

```c
struct Game
{
    int season;
    string opponent;
    string location;
    char W_L_T;
    int UM_score, opponent_score;
};
```
Using structs

struct Game
{
    int season;
    string opponent;
    string location;
    char W_L_T;
    int UM_score,
        opponent_score;
};

- Declared normally

Game game1, game2;
Using structs

Access components using the "dot operator"

```cpp
2006
"Vanderbilt"
Ann Arbor, MI
W
27
7

game1.season = 2006;
game1.opponent="Valderbilt";
```
Use struct members normally

- // add one to game1's UM_score
  game1.UM_score++;


Use struct members normally

- // add one to game1's UM_score
  game1.UM_score++;

- // print a formatted title
  cout << setw(50) << game1.opponent;
Use struct members normally

// add one to game1's UM_score
    game1.UM_score++;

// print a formatted title
    cout << setw(50) << game1.opponent;

- //assign game1 to game2
- // copies ALL members
    game2 = game1;
void printOneGame (Game game) {
    cout << "Season: " << game.season

}
Passing a struct by Value

```cpp
void printOneGame (Game game) {
    cout << "Season: " << game.season
    << " \nOpponent: " << game.opponent
};
```
Passing a struct by Value

```cpp
void printOneGame (Game game) {
    cout << "Season: " << game.season
    << " Opponent: " << game.opponent
    << " Score: " << game.UM_score
    << " " " 
    << game.opponent_score
    << endl;
}
```
void printOneGame (Game game) {
    cout << "Season: " << game.season
    << "\nOpponent: " << game.opponent
    << "\nScore: " << game.UM_score
    << " " << game.opponent_score
    << endl;
    if (game.W_L_T == 'W') {
        cout << "UM ";
    }
}
void printOneGame (Game game) {
    cout << "Season: " << game.season
    << "\nOpponent: " << game.opponent
    << "\nScore: " << game.UM_score
    << " " << game.opponent_score
    << endl;
    if (game.W_L_T == 'W') {
        cout << "UM ";
    } else if (game.W_L_T == 'L') {
        cout << "game.opponent";
    }
}
Passing a struct by Value

```cpp
void printOneGame (Game game) {
    cout << "Season: " << game.season
        << "\nOpponent: " << game.opponent
        << "\nScore: " << game.UM_score
        << " " << game.opponent_score
        << endl;
    if (game.W_L_T == 'W') {
        cout << "UM ";
    } else if (game.W_L_T == 'L') {
        cout << "game.opponent";
    } else {
        cout << "tie";
    }
}
```
call printOneGame

printOneGame (game1);
Passing a struct by Reference

```cpp
void getScores (Game& thisGame)
{
    cout << "Enter UM Score: " ;
    cin >> thisGame.UM_score;
}
```
void getScores (Game& thisGame)
{
    cout << "Enter UM Score: " ;
    cin >> thisGame.UM_score;
    cout << "Enter opponent score: " ;
    cin >> thisGame.opponent_score;
}
call getScores

getScores (game1);
Why use a struct?
stats for 3 seasons

int season1, season2, season3;
Why use a struct? stats for 3 seasons

int season1, season2, season3;
string opponent1, opponent2, opponent3;
Why use a struct? stats for 3 seasons

int season1, season2, season3;
string opponent1, opponent2, opponent3;
string location1, location2, location3;

int UM_score1, UM_score2, UM_score3;
int opponent_score1, opponent_score2, opponent_score3;
Why use a struct?
stats for 3 seasons
int season1, season2, season3;
string opponent1, opponent2, opponent3;
string location1, location2, location3;
char WLT1, WLT2, WLT3;
Why use a struct?
stats for 3 seasons

int season1, season2, season3;
string opponent1, opponent2, opponent3;
string location1, location2, location3;
char WLT1, WLT2, WLT3;
int UM_score1, UM_score2, UM_score3;
Why use a struct? stats for 3 seasons

```c
int season1, season2, season3;
string opponent1, opponent2, opponent3;
string location1, location2, location3;
char WLT1, WLT2, WLT3;
int UM_score1, UM_score2, UM_score3;
int opponent_score1, opponent_score2, opponent_score3;
```
printOneGame

printOneGame(season1, opponent1, location1, WLT1, UM_score1, opponent_score1);
Or ...

Game game1, game2, game3;

printOneGame(game1);
Members of a struct may be structs
Nested (hierarchical) structs

- Members of a struct may be structs
- When to nest structs
  - if nesting improves readability
  - if a structure is used repeatedly inside other structs
  - Beware: too many levels of nesting may impair readability
nested structs

// add date to our game info

struct Date
{
    int month;
    int day;
    int year;
};
struct Game {
    int season;
    Date datePlayed;
    string opponent;
    string location;
    char W_L_T;
    int UM_score, opponent_score;
};
<table>
<thead>
<tr>
<th>season</th>
<th>game3.season</th>
</tr>
</thead>
<tbody>
<tr>
<td>month</td>
<td>game3.location</td>
</tr>
<tr>
<td>day</td>
<td>game3.datePlayed</td>
</tr>
<tr>
<td>year</td>
<td></td>
</tr>
<tr>
<td>opponent</td>
<td></td>
</tr>
<tr>
<td>location</td>
<td></td>
</tr>
<tr>
<td>W_L_T</td>
<td></td>
</tr>
<tr>
<td>UM_score</td>
<td></td>
</tr>
<tr>
<td>opponent_score</td>
<td></td>
</tr>
</tbody>
</table>
Data Structures???

- How to hold data program needs
How to hold data program needs
  - Single variables
  - Homogeneous data layout (arrays)
  - Heterogeneous data layout (structures)
  - Combination
Data Structures???

- Choice of data structure
  - critical in writing a complex program
Data Structures???

- Choice of data structure
  - critical in writing a complex program

- Well planned
  - can make logic simple

- Thrown together
  - can make logic complex