Chapter 11: Structured Data

Hanoi University of Science and Technology

January 5, 2026

Slide contents are mainly based on the following documents:

- Tony Gaddis, Starting out with C++, Six Edition, Pearson, 2009
- Tony Gaddis, Power Points Slides:
  `https://www.umsl.edu/~antognolij/gaddis.html`

Unions

- Similar to a `struct`, but
  - all members share a single memory location, and
  - only one member of the union can be used at a time
- Declared using `union`, otherwise the same as `struct`
- Variables defined as for `struct` variables

Unions

- General format:

```
union union_name
{
  data_type1 member1;
  data_type2 member2;
  data_type3 member3;
  ....
  data_typeN memberN;
};
```

Unions

Program 11-9

```
1  // This program demonstrates a union.
2  #include <iostream>
3  #include <iomanip>
4  using namespace std;
5
6  union PaySource
7  {
8       int hours;          // Hours worked
9       float sales;        // Amount of sales
10 };
11
12 int main()
13 {
14      PaySource employee1;     // Define a union variable
15      char payType;            // To hold the pay type
16      float payRate;           // Hourly pay rate
17      float grossPay;          // Gross pay
18
19      cout << fixed << showpoint << setprecision(2);
20      cout << "This program calculates either hourly wages or\n";
21      cout << "sales commission.\n";
22
23      // Get the pay type, hourly or commission.
24      cout << "Enter H for hourly wages or C for commission: ";
25      cin >> payType;
26
```

Unions

```
27      // Determine the gross pay, depending on the pay type.
28      if (payType == 'H' || payType == 'h')
29      {
30          // This is an hourly paid employee. Get the
31          // pay rate and hours worked.
32          cout << "What is the hourly pay rate? ";
33          cin >> payRate;
34          cout << "How many hours were worked? ";
35          cin >> employee1.hours;
36
37          // Calculate and display the gross pay.
38          grossPay = employee1.hours * payRate;
39          cout << "Gross pay: $" << grossPay << endl;
40      }
41      else if (payType == 'C' || payType == 'c')
42      {
43          // This is a commission-paid employee. Get the
44          // amount of sales.
45          cout << "What are the total sales for this employee? ";
46          cin >> employee1.sales;
47
48          // Calculate and display the gross pay.
49          grossPay = employee1.sales * 0.10;
50          cout << "Gross pay: $" << grossPay << endl;
51      }
```

(program continues)

Program 11-9 *(continued)*

```
52        else
53        {
54             // The user made an invalid selection.
55             cout << payType << " is not a valid selection.\n";
56        }
57        return 0;
58   }
```

Program Output with Example Input Shown in Bold

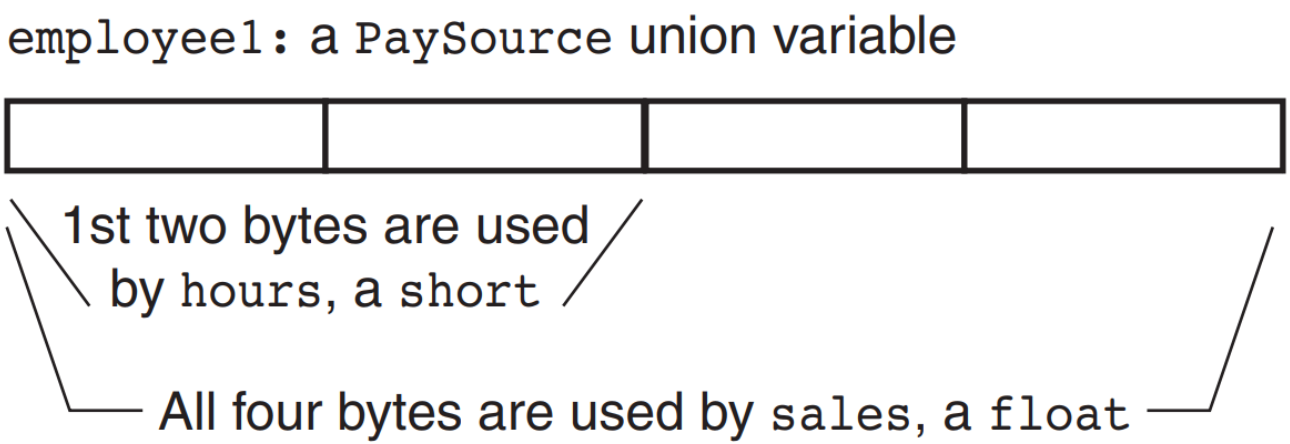
```
This program calculates either hourly wages or
sales commission.
Enter H for hourly wages or C for commission: C [Enter]
What are the total sales for this employee? 5000 [Enter]
Gross pay: $500.00
```

Program Output with Different Example Input Shown in Bold

```
This program calculates either hourly wages or
sales commission.
Enter H for hourly wages or C for commission: H [Enter]
What is the hourly pay rate? 20 [Enter]
How many hours were worked? 40 [Enter]
Gross pay: $800.00
```

Unions

struct	**union**
The size of a `struct` is usually greater than or equal to the sum of the sizes of all its members.	The size of a `union` is equal to the size of its largest attribute.
Can store multiple values for members simultaneously.	Only one member can store a value at a time.



Anonymous Union

- A union itself has no name:
  ```
  union { ... };
  ```
- Can refer to members directly without dot operator
- Must use `static` if declared outside of a function

Enumerated Data Types

- Problem:

```
const int MONDAY = 0;
const int TUESDAY = 1;
const int WEDNESDAY = 2;
const int THURSDAY = 3;
const int FRIDAY = 4;
```

  We need at least 5 variables ... Should group constants with equivalent meanings into separate groups of constants

- An enumerated data type is a programmer-defined data type. It consists of values known as **enumerators**, which represent integer constants.
- General format:

```
enum <name_of_enumeration>
{     //list all of values inside this block
  //each enumerator is separated by a comma
};
```

Enumerated Data Types

- Example:

```
enum Day { MONDAY, TUESDAY,
           WEDNESDAY, THURSDAY, FRIDAY };
```

- The identifiers `MONDAY`, `TUESDAY`, `WEDNESDAY`, `THURSDAY`, and `FRIDAY`, which are listed inside the braces, are enumerators. They represent the values that belong to the `Day` data type.
- Note that the enumerators are not strings, so they aren't enclosed in quotes. They are identifiers.

Enumerated Data Types

- Once you have created an enumerated data type in your program, you can define variables of that type. Example:

```
Day workDay;
```

- This statement defines `workDay` as a variable of the `Day` type.
- We may assign any of the enumerators `MONDAY`, `TUESDAY`, `WEDNESDAY`, `THURSDAY`, `FRIDAY` to a variable of the Day type. Example:

```
workDay = WEDNESDAY;
```

Enumerated Data Types

- So, what is an enumerator?
- Think of it as an integer named constant
- Internally, the compiler assigns integer values to the enumerators, beginning at 0.
- `enum Day { MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY };`

  In memory...

  `MONDAY = 0`
  `TUESDAY = 1`
  `WEDNESDAY = 2`
  `THURSDAY = 3`
  `FRIDAY = 4`

Enumerated Data Types

- Using the Day declaration, the following code...

```
cout << MONDAY << " "
     << WEDNESDAY << " "
     << FRIDAY << endl;
```

...will produce this output: `0 2 4`

Assigning an integer to an enum Variable

- You cannot directly assign an integer value to an enum variable. This will not work:

```
workDay = 3; // Error!
```

- Instead, you must cast the integer:

```
workDay = static_cast<Day>(3);
```

Assigning an integer to an `enum` Variable

- You CAN assign an enumerator to an int variable. For example:

```
int x;
x = THURSDAY;
```

- This code assigns 3 to `x`.

Comparing Enumerator Values

- Enumerator values can be compared using the relational operators. For example, using the Day data type the following code will display the message "Friday is greater than Monday."

```
if (FRIDAY > MONDAY)
{
  cout << "Friday is greater than Monday.\n";
}
```

Comparing Enumerator Values

Program 11-11

```
1  // This program demonstrates an enumerated data type.
2  #include <iostream>
3  #include <iomanip>
4  using namespace std;
5
6  enum Day { MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY };
7
8  int main()
9  {
10     const int NUM_DAYS = 5;     // The number of days
11     double sales[NUM_DAYS];     // To hold sales for each day
12     double total = 0.0;         // Accumulator
13     int index;                  // Loop counter
14
15     // Get the sales for each day.
16     for (index = MONDAY; index <= FRIDAY; index++)
17     {
18         cout << "Enter the sales for day "
19              << index << ": ";
20         cin >> sales[index];
21     }
```

Comparing Enumerator Values

```
22
23       // Calculate the total sales.
24       for (index = MONDAY; index <= FRIDAY; index++)
25          total += sales[index];
26
27       // Display the total.
28       cout << "The total sales are $" << setprecision(2)
29            << fixed << total << endl;
30
31       return 0;
32    }
```

Program Output with Example Input Shown in Bold

```
Enter the sales for day 0: 1525.00 [Enter]
Enter the sales for day 1: 1896.50 [Enter]
Enter the sales for day 2: 1975.63 [Enter]
Enter the sales for day 3: 1678.33 [Enter]
Enter the sales for day 4: 1498.52 [Enter]
The total sales are $8573.98
```

Anonymous Enumerated Types

- An `anonymous enumerated` type is simply one that does not have a name. For example:
  `enum { MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY };`

Using Math Operators with enum Variables

- You can run into problems when trying to perform math operations with enum variables. For example:

```
Day day1, day2; // Define two Day variables.
day1 = TUESDAY; // Assign TUESDAY to day1.
day2 = day1 + 1;// ERROR! Will not work!
```

- The third statement will not work because the expression day1 + 1 results in the integer value 2, and you cannot store an int in an enum variable.

Using Math Operators with enum Variables

- You can fix this by using a cast to explicitly convert the result to Day, as shown here:

```
// This will work.
day2 = static_cast<Day>(day1 + 1);
```

Using an enum Variable to Step through an Array's Elements

- Because enumerators are stored in memory as integers, you can use them as array subscripts. For example:

```
enum Day { MONDAY, TUESDAY,
           WEDNESDAY, THURSDAY, FRIDAY };
const int NUM_DAYS = 5;
double sales[NUM_DAYS];
sales[MONDAY] = 1525.0;
sales[TUESDAY] = 1896.5;
sales[WEDNESDAY] = 1975.63;
sales[THURSDAY] = 1678.33;
sales[FRIDAY] = 1498.52;
```

Using an enum Variable to Step through an Array's Elements

- Remember, though, you cannot use the ++ operator on an enum variable. So, the following loop will NOT work.

```
Day workDay;  // Define a Day variable
// ERROR!!! This code will NOT work.
for (workDay = MONDAY; workDay <= FRIDAY; workDay++)
{
   cout << "Enter the sales for day "
        << workDay << ": ";
   cin >> sales[workDay];
}
```

Using an enum Variable to Step through an Array's Elements

- You must rewrite the loop's update expression using a cast instead of ++:

```
for (workDay = MONDAY; workDay <= FRIDAY;
                workDay = static_cast<Day>(workDay + 1))
{
   cout << "Enter the sales for day "
        << workDay << ": ";
   cin >> sales[workDay];
}
```

Using an enum Variable to Step through an Array's Elements

Program 11-12

```
1  // This program demonstrates an enumerated data type.
2  #include <iostream>
3  #include <iomanip>
4  using namespace std;
5
6  enum Day { MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY };
7
8  int main()
9  {
10     const int NUM_DAYS = 5;     // The number of days
11     double sales[NUM_DAYS];     // To hold sales for each day
12     double total = 0.0;         // Accumulator
13     Day workDay;                // Loop counter
14
```

(program continues)

Using an enum Variable to Step through an Array's Elements

Program 11-12 *(continued)*

```
15      // Get the sales for each day.
16      for (workDay = MONDAY; workDay <= FRIDAY;
17                             workDay = static_cast<Day>(workDay + 1))
18      {
19          cout << "Enter the sales for day "
20               << workDay << ": ";
21          cin >> sales[workDay];
22      }
23
24      // Calculate the total sales.
25      for (workDay = MONDAY; workDay <= FRIDAY;
26                             workDay = static_cast<Day>(workDay + 1))
27          total += sales[workDay];
28
29      // Display the total.
30      cout << "The total sales are $" << setprecision(2)
31           << fixed << total << endl;
32
33      return 0;
34  }
```

Program Output with Example Input Shown in Bold

```
Enter the sales for day 0: 1525.00 [Enter]
Enter the sales for day 1: 1896.50 [Enter]
Enter the sales for day 2: 1975.63 [Enter]
Enter the sales for day 3: 1678.33 [Enter]
Enter the sales for day 4: 1498.52 [Enter]
The total sales are $8573.98
```

Specifying Integer Values for Enumerators

- By default, the enumerators in an enumerated data type are assigned the integer values 0, 1, 2, and so forth.
- You can specify the values to be assigned. Example:

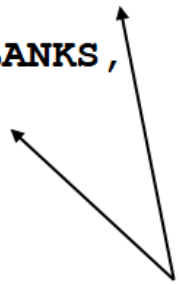
```
enum Water  FREEZING = 32, BOILING = 212 ;
```

Using an enum Variable to Step through an Array's Elements

- Enumerators must be unique within the same scope. For example, an error will result if both of the following enumerated types are declared within the same scope:

```
enum Presidents { MCKINLEY, ROOSEVELT, TAFT };

enum VicePresidents { ROOSEVELT, FAIRBANKS,
                      SHERMAN };
```



ROOSEVELT is declared twice.

Declaring the Type and Defining the Variables in One Statement

- You can declare an enumerated data type and define one or more variables of the type in the same statement. For example:
  `enum Car { PORSCHE, FERRARI, JAGUAR } sportsCar;`
- This code declares the `Car` data type and defines a variable named `sportsCar`.

Exercises

- Programming Challenges: 11, 15