BITG 1233: STRUCTURED DATA

LECTURE 11
(Sem 2, 17/18)
LEARNING OUTCOMES

At the end of this lecture, you should be able to:

1. Describe the concept of structure.
2. Describe the difference between array and structure.
3. Describe the concept of structure containing array and array of structure.
4. Use structures in program.
5. Use structures as Function Parameters.
Combining Data into Structures

- **Structure** – is a collection of related elements, possibly of different types, having a single name.
- C++ construct that allows multiple variables to be grouped together.
- Each variable is an element in a structure.
- Element of a structure is called a field or member.

- The **difference** between an array and a structure is that all elements in an array must be of the same type, while the elements in a structure may be of the same or different types.
Structure

• The first example (fraction), has two fields, both of which are integers.
• The second example (student), has three fields, an integer number, a character array and a floating point number.
To declare a **structure type**, use the general format as follows:

```c
struct <structureName>
{
    type1   field1;
    type2   field2;
    . . .
};
```

**Structure Declaration Notes:**

- Must have `;` after closing `}
- Multiple fields of same type can be in comma-separated list:
  
  Example: `char name[25], address[30];`
- To define a variable of a structure type, use the **structure name** as the variable’s type.
• Example:

**Structure declaration:**

```c
struct STUDENT {
    int studentID;
    char name[30];
    short yearInSchool;
    double gpa;
};
```

**It’s variable declaration:**

```c
STUDENT bill;
```

Or,

```c
struct STUDENT bill;
```

**Example:**

- **Structure declaration:**
  ```c
  struct STUDENT {
  int studentID;
  char name[30];
  short yearInSchool;
  double gpa;
  }
  ```

- **It’s variable declaration:**
  ```c
  STUDENT bill;
  ```
Structure Initialization

- A structure can be initialized.

- The **rules of structure initialization** are similar to the rules of array initialization:
  
  (1) the initializers are **enclosed in braces and separated by commas**;
  
  (2) the initializers must **match their corresponding types** in the structure declaration;
  
  (3) if a **nested structure** is used, the nested initializers must be enclosed in their own set of braces.
Structure Initialization

Example:

```c
struct SAMPLE
{
    int    x;
    int    y;
    float  t;
    char   u;
};

SAMPLE sam1 = { 2, 5, 3.2, 'A' };
SAMPLE sam2 = { 7, 3 };
```
We can read data into and write data from structure members just as we can from individual variables.

Use the **dot (.) operator** to refer to **members of struct variables**.

For example the value for the field of the sample structure can be read from the keyboard and placed in `sam2` using the input statement below. Suppose, the data entered are 7, 3, 0, ’R’.

```cpp
cin >> sam2.x >> sam2.y >> sam2.t >> sam2.u;
```
Structure Operation

• The structure is an entity that can be treated as a whole.
• However, only one operation (assignment) is allowed on the whole structure itself. In other words, a structure can only be copied to another structure of the same type using the assignment operator.
• Example:

![Diagram showing structure operation]

Before:

```
<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>t</th>
<th>u</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>3</td>
<td>0.0</td>
<td>R</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>t</th>
<th>u</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>5</td>
<td>3.2</td>
<td>A</td>
</tr>
</tbody>
</table>
```

```
sam2 = sam1;
```

After:

```
<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>t</th>
<th>u</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>5</td>
<td>3.2</td>
<td>A</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>t</th>
<th>u</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>5</td>
<td>3.2</td>
<td>A</td>
</tr>
</tbody>
</table>
```

Displaying a `struct` Variable

To display the contents of a `struct` variable, must display each field separately, using the dot operator:

```cpp
cout << bill; // won’t work
cout << bill.studentID << endl;
cout << bill.name << endl;
cout << bill.yearInSchool;
cout << " " << bill.gpa;
```
Comparing `struct` Variables

- Cannot compare `struct` variables directly:
  ```c
  if (bill == william) // won't work
  ```

- Instead, must compare on a field/member basis:
  ```c
  if (bill.studentID == william.studentID)
  ```
struct STUDDATA
{
    int id;
    char name[20];
    float gradePoint;
};

void main( )
{
    STUDDATA studBITG1113;

    cout << "Please enter your id";
    cin >> studBITG1113.id;
    cout << "Please enter your name";
    cin.getline(studBITG1113.name,19);
    cout << "Please enter your gradePoint";
    cin >> studBITG1113.gradePoint;

    cout << "Your id is :" << studBITG1113.id;
    cout << "Your name is :" << studBITG1113.name;
    cout << "Your grade point is :" << studBITG1113.gradePoint;

    if(studBITG1113.gradePoint > 3.5)
    
        cout<<"Excellent!"
}
Nested Structure

• We can have structures as members of a structure.

• When a structure includes another structure, it is a nested structure.

• For example, we can have a structure called STAMP that stores the date and the time.

• The DATE is in turn a structure that stores the month, day and year.

• The TIME is also a structure, one that stores the hour, minute and second.
• Declaration of the structure called **STAMP** is as follows:

```c
struct DATE
{
    int month;
    int day;
    int year;
};
struct TIME
{
    int hour;
    int min;
    int sec;
};
struct STAMP
{
    DATE date;
    TIME time;
};

STAMP stamp;
```
• It is possible to nest the same structure type more than once in a declaration.

• Example:

```c
struct JOB {
    STAMP startTime;
    STAMP endTime;
};
JOB job;
```
• When accessing a nested structure, we must include each level from the **highest (stamp)** to the component being referenced.
Structure Containing Array

- Structures can have one or more arrays as members.

```c
/*Global declarations */
struct PUPIL
{
    char name[26];
    int midterm[3];
    int final;
};

PUPIL student;
```
struct STUDDATA
{
    char name[20];
    float test[3];
    float ass[5];
    float quiz[2];
    float final;
    float total;
    float project;
};

void main( )
{
    STUDDATA studBITG1113;
    float totTest=0, totAss=0, totQuiz=0;

    cout << "Please enter your name : ";
    cin.getline(studBITG1113.name,19);

    for( int i = 0; i < 3; i++ )
    {
        cout << "Please enter the score for test : " << i+1;
        cin >> studBITG1113.test[i];
        totTest += studBITG1113.test[i];
    }

    // Program continues to the next slide......
for(i=0; i<5; i++)
{
    cout << "Please enter the score for assignment"<<i+1<<" : ";
    cin >> studBITG1113.ass[i];
    totAss += studBITG1113.ass[i];
}

for(i=0; i<2; i++)
{
    cout << "Please enter the score for quiz"<<i+1<<" : ";
    cin >> studBITG1113.quiz[i];
    totQuiz += studBITG1113.quiz[i];
}

cout << "Please enter the score for final : ";
cin >> studBITG1113.final;

cout << "Please enter the score for project : ";
cin >> studBITG1113.project;

studBITG1113.total = totTest + totAss + totQuiz +
    studBITG1113.final + studBITG1113.project;

cout << "Your score for this subject is : " << studBITG1113.total;
} // end of main()
Output:

Please enter your name: Muhammad Ammar
Please enter the score for test 1: 8
Please enter the score for test 2: 9
Please enter the score for test 3: 13
Please enter the score for assignment 1: 2.5
Please enter the score for assignment 2: 2.6
Please enter the score for assignment 3: 2.7
Please enter the score for assignment 4: 3
Please enter the score for assignment 5: 3
Please enter the score for quiz 1: 2.4
Please enter the score for quiz 2: 2.2
Please enter the score for final: 25
Please enter the score for project: 13
Your score for this subject is: 86.4
• As a programmer, you will encounter many situations that require you to create an array of structures.

• By putting the data in an array, we can quickly and easily work with the data.

• Example array of structures might look like:

<table>
<thead>
<tr>
<th>stuAry[0]</th>
<th>...</th>
<th>stuAry[1]</th>
<th>...</th>
<th>stuAry[2]</th>
<th>...</th>
<th>...</th>
<th>stuAry[49]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

stuAry
**Example: Array of Structure**

```c
struct PELAJAR
{
    int id;
    char name[31];
    float project_mark;
    int test_mark;
    int final_mark;
    char gred;
};

PELAJAR rekod_pelajar[3];

OR with initialization:

```c
struct pelajar rekod_pelajar[] = {
{1342, "Zulaiha Ismail", 10.2, 10, 20, 'F'},
{1343, "Aina Ahmad", 51.4, 60, 60, 'C'},
{1344, "Maria Musa", 90.0, 99, 99, 'A'}
};
```
Example : Array of Structure

- To print the elements in rekod_pelajar:

```cpp
for(i=0; i<3; i++)
{
    cout << rekod_pelajar[i].id << endl;
    cout << rekod_pelajar[i].name << endl;
    cout << rekod_pelajar[i].project_mark << endl;
    cout << rekod_pelajar[i].test_mark << endl;
    cout << rekod_pelajar[i].final_mark << endl;
    cout << rekod_pelajar[i].gred << endl;
}
```
struct STUDDATA
{
    char name[20];
    float test[3];
    float ass[5];
    float quiz[2];
    float final;
    float total;
    float project;
};

int main( )
{
    STUDDATA studBITG1113[50];
    float totTest=0, totAss=0, totQuiz=0;

    for( int i=0; i<50; i++ )
    {
        cout << "Enter your name : ";
        cin.getline(studBITG1113[i].name,19);

        for( int t=0; t<3; t++ )
        {
            cout << "Enter the score for test : " << i+1;
            cin >> studBITG1113[i].test[t];
            totTest += studBITG1113[i].test[t];
        }
    }
}

*Program continues to the next slide......


```cpp
for( int a=0; a<5; a++ )
{
    cout << "\nPlease enter the score for assignment"<<i+1<<" : ";
    cin >> studBITG1113[i].ass[a];
    totAss += studBITG1113[i].ass[a];
}

for( int q=0; q<2; q++ )
{
    cout << "\nPlease enter the score for quiz"<<i+1 <<" : ";
    cin >> studBITG1113[i].quiz[q];
    totQuiz += studBITG1113[i].quiz[q];
}

cout << "\nPlease enter the score for final : ";
cin >> studBITG1113[i].final;

studBITG1113[i].total = totTest + totAss + totQuiz +
    studBITG1113[i].final + studBITG1113[i].project;
cout <<"\nYour score for this subject is : " <<studBITG1113[i].total;
}//for i
return 0;
} //end of main()
```
May pass members of \texttt{struct} variables to functions:

\begin{verbatim}
computeGPA(stu.gpa);
\end{verbatim}

May pass entire \texttt{struct} variables to functions:

\begin{verbatim}
showData(stu);
\end{verbatim}

Can use reference parameter if function needs to modify contents of structure variable.
A Code Snippet: struct as function parameter

```
const int DESC_SIZE = 50; // Array size

struct InventoryItem
{
    int partNum; // Part number
    char description[DESC_SIZE]; // Item description
    int onHand; // Units on hand
    double price; // Unit price
};

void showItem(InventoryItem p)
{
    cout << fixed << showpoint << setprecision(2);
    cout << "Part Number: " << p.partNum << endl;
    cout << "Description: " << p.description << endl;
    cout << "Units On Hand: " << p.onHand << endl;
    cout << "Price: $" << p.price << endl;
}
```
Structures as Function Parameters - Notes

- Using value parameter for structure can slow down a program, waste space.
- Using a \textbf{reference parameter} will speed up program, but function may change data in structure.
- Using a \textbf{const} reference parameter allows read-only access to reference parameter, does not waste space, speed.
Revised showItem Function

```cpp
void showItem(const InventoryItem &p) {
    cout << fixed << showpoint << setprecision(2);
    cout << "Part Number: " << p.partNum << endl;
    cout << "Description: " << p.description << endl;
    cout << "Units On Hand: " << p.onHand << endl;
    cout << "Price: $" << p.price << endl;
}
```
Returning a Structure from a Function

- Function **can return a struct:**

  ```c
  STUDENT getStudentData();  // prototype
  stu1 = getStudentData();   // function call
  ```

- Function must define a **local structure**
  - for internal use
  - for use with **return statement**
Returning a Structure from a Function - Example

```cpp
STUDENT getStudentData()
{
    STUDENT tempStu;
    cin >> tempStu.studentID;
    getline(cin, tempStu.pData.name);
    getline(cin, tempStu.pData.address);
    getline(cin, tempStu.pData.city);
    cin >> tempStu.yearInSchool;
    cin >> tempStu.gpa;
    return tempStu;
}
```
Program 11- 4 : pg1

// This program uses a function to return a structure.

#include <iostream>
#include <iomanip>
#include <cmath> // For the pow function
using namespace std;

// Constant for Pi.
const double PI = 3.14159;

// Structure declaration
struct Circle
{
    double radius;       // A circle's radius
    double diameter;     // A circle's diameter
    double area;         // A circle's area
};

// Function prototype
Circle getInfo();

int main()
{
    Circle c;     // Define a structure variable
// Get data about the circle.
c = getInfo();

// Calculate the circle's area.
c.area = PI * pow(c.radius, 2.0);

// Display the circle data.
cout << "The radius and area of the circle are:\n";
cout << fixed << setprecision(2);
cout << "Radius: " << c.radius << endl;
cout << "Area: " << c.area << endl;
return 0;
Program 11-4: Continued (pg3)

```
40 //***************************************************************************************
41 // Definition of function getInfo. This function uses a local * 
42 // variable, tempCircle, which is a circle structure. The user * 
43 // enters the diameter of the circle, which is stored in * 
44 // tempCircle.diameter. The function then calculates the radius * 
45 // which is stored in tempCircle.radius. tempCircle is then * 
46 // returned from the function. 
47 //***************************************************************************************
48 Circle getInfo()
49 {
50   Circle tempCircle; // Temporary structure variable
51
52   // Store circle data in the temporary variable.
53   cout << "Enter the diameter of a circle: ";
54   cin >> tempCircle.diameter;
55   tempCircle.radius = tempCircle.diameter / 2.0;
56
57   // Return the temporary variable.
58   return tempCircle;
59 }
```

**Program Output with Example Input Shown in Bold**

Enter the diameter of a circle: 10 [Enter]
The radius and area of the circle are:
Radius: 5.00
Area: 78.54