CIS 129
Advanced Computer Programming

Chapter 7: User Defined Datatypes: Structure

Mr. Horence Chan

User Defined Datatypes

- In a variable, one can store a single value of a particular type.
  - e.g. `int a;`
  - Allows you to store a single value of type integer in the variable `a`.
- In an array, one can store many values but again of the same type.
  - e.g. `int a[10];`
  - Allows you to store 10 values of type integer in the array `a`.
- In C++, user-defined datatypes enable you to store many values of **different** datatypes and group them into a **single** unit.
- The two main types of these user defined datatypes are **structures** and **classes**.

Structure

```
_________ Student{
        string firstName;
        string lastName;
        char courseGrade;
        int assignmentScore;
        int examScore;
        double GPA;
};
```

- Structure (`struct`): a collection of a fixed number of components in which the components are accessed by name. The components may be of **different** types.
- The components of a `struct` are called the _____ of the `struct`

Structure Members

```
struct Student{
        string firstName;
        string lastName;
        char courseGrade;
        int assignmentScore;
        int examScore;
        double GPA;
};
```

- Defines a struct "`Student`" with six members.
- Members `firstName` and _____ are of type string
- Members `courseGrade` is of type _____
- Members `assignmentScore` and `examScore` are of type int
- Members `GPA` is of type double

Structure Variable

```
struct Student{
        string firstName;
        string lastName;
        char courseGrade;
        int assignmentScore;
        int examScore;
        double GPA;
};
int main(){
        Student student1;
        student1.GPA = 0.0;
        student1.firstName = "Peter";
        student1.lastName = "Parker";

         ...
}
```

Struct: Student	student1
firstName	
lastName	
courseGrade	
assignmentScore	
examScore	
GPA	

- Declare `student1` to be a **variable** of type `Student`
- Initialize the member `GPA` of `student1` to 0.0
- Store "`Peter`" in the member `firstName` of `student1`
- Store "`Parker`" in the member `lastName` of `student1`

Structure and Input

```
struct Student{
        string firstName;
        string lastName;
        char courseGrade;
        int assignmentScore;
        int examScore;
        double GPA;
};
int main(){
...
        int score;
        cin >> student1.assignmentScore;
        cin >> student1.examScore;
        score = (student1.assignmentScore +
                student1. examScore) / 2 ;
...
}
```

Struct: Student	student1
firstName	Peter
lastName	Parker
courseGrade	
assignmentScore	
examScore	
GPA	0.0

- Read the input from user and store to `assignmentScore` and `examScore` of `student1`
- Example:

  90

  75

```
struct Student{
...
        char courseGrade;
        double GPA;
};
int main(){
...
if(score >= 90){
        student1.courseGrade = 'A';
        student1.GPA = 4.0;}
else if(score >= 80){
        student1.courseGrade = 'B';
        student1.GPA = 3.0;}
else if(score >= 70){
        student1.courseGrade = 'C';
        student1.GPA = 2.0;}
else if(score >= 60){
        student1.courseGrade = 'D';
        student1.GPA = 1.0;}
else
        student1.courseGrade = 'F';
...}
```

Struct: Student	student1
firstName	Peter
lastName	Parker
courseGrade	
assignmentScore	90
examScore	75
GPA	0.0

- Determines the course grade and GPA and store to `courseGrade` and `GPA` of `student1`

Structure

```
struct Student{
        string firstName;
        string lastName;
        char courseGrade;
        int assignmentScore;
        int examScore;
        double GPA;
};
int main(){
...
        Student student2;
        student2 = {"Mary", "Jane", 'A', 99, 97, 4.0};

...
}
```

Struct: Student	student1	student2
firstName	Peter	
lastName	Parker	
courseGrade	B	
assignmentScore	90	
examScore	75	
GPA	3.0	

- We can also add the information to the structure in this format
- It follows the alignment of the structure
- Note: **all** the entries must be filled

```
struct Student{
        string firstName;
        string lastName;
        char courseGrade;
        int assignmentScore;
        int examScore;
        double GPA;
};
int main(){
...
student1.GPA += 0.3;
cout << student1.firstName << " " << student1.lastName << endl;
cout << "Grade: " << student1.courseGrade << endl;
cout << "GPA: " << student1.GPA << endl;
cout << student2.firstName << " " << student2.lastName << endl;
cout << "Grade: " << student2.courseGrade << endl;
cout << "GPA: " << student2.GPA << endl;
return 0;
}
```

Struct: Student	student1	student2
firstName	Peter	Mary
lastName	Parker	Jane
courseGrade	B	A
assignmentScore	90	99
examScore	75	97
GPA		4.0

- We can increment the value of `integers` and `double` in `structure`
- Note we cannot do operations on an _____ structure variable
- e.g.:`cin>>student1;` or `cout<<student2;` are not valid.

```
struct Student{
        string firstName;
        string lastName;
        char courseGrade;
        int assignmentScore;
        int examScore;
        double GPA;
};
int main(){
...
student1.GPA += 0.3;
cout << student1.firstName << " " << student1.lastName << endl;
cout << "Grade: " << student1.courseGrade << endl;
cout << "GPA: " << student1.GPA << endl;
cout << student2.firstName << " " << student2.lastName << endl;
cout << "Grade: " << student2.courseGrade << endl;
cout << "GPA: " << student2.GPA << endl;

return 0;
}
```

Input:

90

75

Output:

Peter Parker

Grade: _____

GPA: _____

Mary Jane

Grade: _____

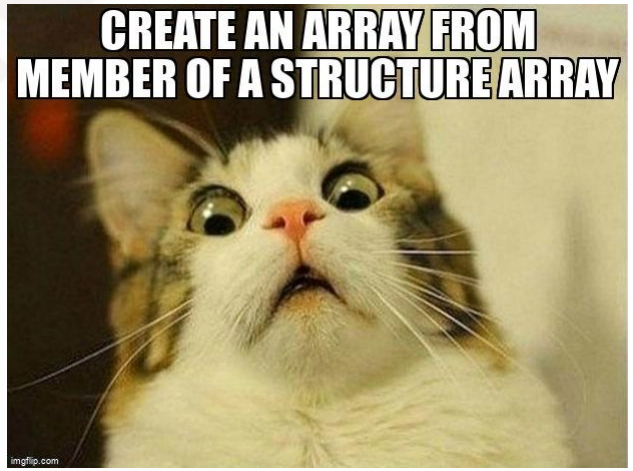
GPA: 4

Structure and Array

```
struct Student{
        string Name;
        int Scores[2];
        double aveScore;
};
int main(){

        Student student[3];
        student[0].Name = "Parker";
        student[1].Name = "Jane";
        student[2].Name = "Hogan";



...
}
```

Struct: Student	student[0]	student[1]	student[2]
Name			
Scores[2]			
aveScore			

- If we have a lot of `struct` variables, we can use an array to store it.



Structure and File Input

```
struct Student{
        string Name;
        int Scores[2];
        double aveScore;
};
int main(){
        Student student[3];
        student[0].Name = "Parker";
        student[1].Name = "Jane";
        student[2].Name = "Hogan";


...
}
```

- Given a file store the student marks in semester 1 and semester 2 programming courses

Jane	2	89
Hogan	1	34
Jane	1	93
Parker	2	78
Parker	1	52
Hogan	2	59

- We need to read this file and put the data to the structure

Structure and File Input

```
struct Student{
        string Name;
        int Scores[2];
        double aveScore;
};
void readMarks(ifstream____ inp, Student ss[], int
NoofStudent);
int main(){
...
        ifstream infile;
        infile.open("marks.txt");
        readMarks(_________, _________, _________);
...
}
```

- Open the file “marks.txt”
- Create a function `readMarks` to read the files and put the data to the structure
- Note: `ifstream` datatype must be **reference (&)**

Structure and Function

```
struct Student{
        string Name;
        int Scores[2];
        double aveScore;
};
void readMarks(ifstream& inp, Student ss[], int NoofStudent){
        int index, semester;
        string ID;
        double mark;
        for (index = 0; index < NoofStudent; index++){
                for (int i = 0; i < 2; i++){
                        ss[index].Scores[i] = 0;
                }
        }
...
}
```

Struct: Student		student[0]	student[1]	student[2]
Name		Parker	Jane	Hogan
Scores	Scores[0]			
	Scores[1]			
aveScore				

- Initialize the scores of each student on each semester to _____.
- It is a good habit to do initialization in the beginning, as it make sure everything is ready for calculation

Structure and Function

Struct: Student		student[0]	student[1]	student[2]
Name		Parker	Jane	Hogan
Scores	Scores[0]			
	Scores[1]			
aveScore				

```
struct Student{
        string Name;
        int Scores[2];
        double aveScore;
};
void readMarks(ifstream& inp, Student ss[], int NoofStudent){
...
        inp >> ID;
    while (_______){
        inp >> semester >> mark;
        for (index = 0; index < NoofStudent; index++){
            if (ID == ss[index].Name){
                break;}}


        ____________________________________

        inp >> ID;
    }
}
```

Jane	2	89
Hogan	1	34
…	…	…

- Read first student name
- While the input is not **null**
- Read semester and marks
- Use **linear search** to find the index of student name
- Increment the marks to the correct student and semester
- Read next student name

Structure and File Output

```
struct Student{
        string Name;
        int Scores[2];
        double aveScore;
};
void writeMarks(ofstream_____ out, Student ss[], int NoofStudent);
int main(){
...
        ofstream outfile;
        outfile.open("marks_output.out");
        writeMarks(_________, _________, _________);
...
}
```

- Create a file “marks_output.out”
- Create a function `writeMarks` to write the data in the structure to a file
- Note: `ofstream` datatype must be **reference (&)**

Structure and Output

```
struct Student{
        string Name;
        int Scores[2];
        double aveScore;
};
void writeMarks(ofstream& out, Student ss[], int NoofStudent){
 int index;
 int semester;
 out << setw(15) << setfill(' ') << left << "Name:";
 for (index = 0; index < NoofStudent; index++){
       out << setw(10) << setfill(' ') << left << ss[index].Name;}
 out << endl << setw(15) << setfill(' ') << left << "Sem1 Score:";
 for (index = 0; index < NoofStudent; index++){
       out << setw(10) << setfill(' ') << left << ss[index]. Scores[0];}
 out << endl << setw(15) << setfill(' ') << left << "Sem2 Score:";
 for (index = 0; index < NoofStudent; index++){
       out << setw(10) << setfill(' ') << left << ss[index]. Scores[1];}
}
```

- Output file

Name: Parker Jane Hogan

Sem1 Score:

Sem2 Score:

Structure (Find average)

```
struct Student{
        string Name;
        int Scores[2];
        double aveScore;
};
int main(){
...
        for (int j = 0; j < 3; j++){
                student[j].aveScore = ______;
                }
        for (int j = 0; j < 3; j++){
                for (int i = 0; i < 2; i++){
                        ____________________________________________
                }
                student[j].aveScore /= 2.0;
        }
}
```

Struct: Student		student[0]	student[1]	student[2]
Name		Parker	Jane	Hogan
Scores	Scores[0]	52	93	34
	Scores[1]	78	89	59
aveScore				

- Initialize the average scores of each student on each semester to 0.
- Find the average score of each student

Structure (Find average)

```
struct Student{
        string Name;
        int Scores[2];
        double aveScore;
};
void writeMarks(ofstream& out, Student ss[], int NoofStudent){
 ...
out << endl << setw(15) << setfill(' ') << left << "Average:";
for (index = 0; index < NoofStudent; index++){
        out << setw(10) << setfill(' ') << left << ss[index]. aveScore;
        }
}
```

- Output file

Name:	Parker	Jane	Hogan
Sem1 Score:	52	93	34
Sem2 Score:	78	89	59
Average:			

Structure (Find highest)

```
struct Student{
        string Name;
        int Scores[2];
        double aveScore;
};
int main(){
...
        string highststudent;
        int highestmark = 0;
        for (int k = 0; k < 3; k++){
                _____________________________________________
                _____________________________________________
                _____________________________________________
                        }
        }
outfile << endl << endl << "Highest Marks: " << highestmark << ", " << highststudent;
}
```

Struct: Student		student[0]	student[1]	student[2]
Name		Parker	Jane	Hogan
Scores	Scores[0]	52	93	34
	Scores[1]	78	89	59
aveScore				

- Find the student who got the highest average marks
- Output

Highest Marks:_____