Misr International University (MIU)

Faculty of Computer Science

Computer Science Program

CSC 105: Introduction to Programming & Problem Solving

Dr. Ayman Ezzat and Dr. Ashraf AbdelRaouf

Lecture 6

Records (structs)

Objectives

In this Lecture, you will:

- Learn about records (`structs`)
- Examine various operations on a `struct`
- Explore ways to manipulate data using a `struct`
- Learn about the relationship between a `struct` and functions
- Discover how arrays are used in a `struct`
- Learn how to create an array of `struct` items

Records (structs)

- `struct`: collection of a fixed number of components (members), accessed by name
  - Members may be of different types
- Syntax:

```
struct structName
{
    dataType1 identifier1;
    dataType2 identifier2;
      .
      .
      .
    dataTypen identifiern;
};
```

Records (structs) (continued)

- A `struct` is a definition, not a declaration

```
struct studentType
{
    string firstName;
    string lastName;
    char courseGrade;
    int testScore;
    int programmingScore;
    double GPA;
};

    //variable declaration
studentType newStudent;
studentType student;
```

```
struct employeeType
{
    string firstName;
    string lastName;
    string address1;
    string address2;
    double salary;
    string deptID;
};
```

Records (structs) (continued)

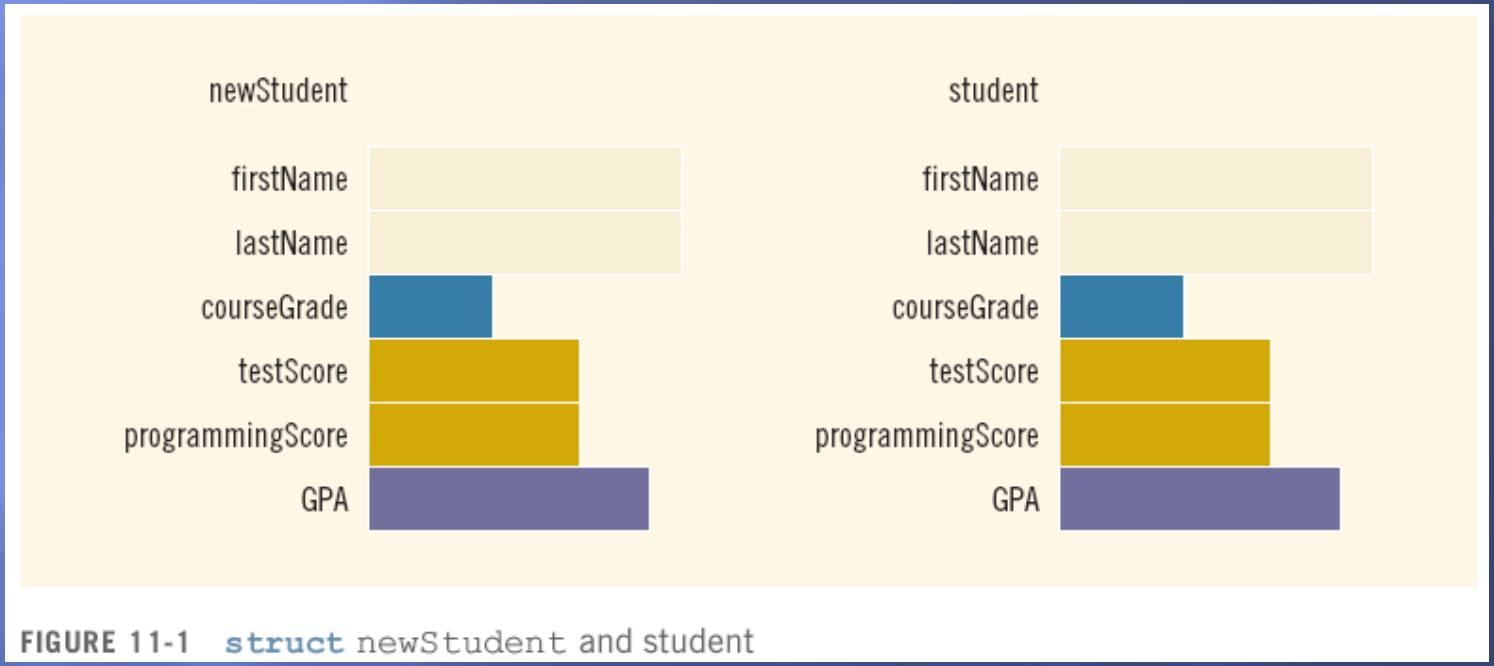

FIGURE 11-1 `struct newStudent` and student

Accessing `struct` Members

- The syntax for accessing a `struct` member is:

```
structVariableName.memberName
```

- The dot (.) is an operator, called the member access operator

Accessing `struct` Members (continued)

- To initialize the members of `newStudent`:

```
newStudent.GPA = 0.0;
newStudent.firstName = "John";
newStudent.lastName = "Brown";
```

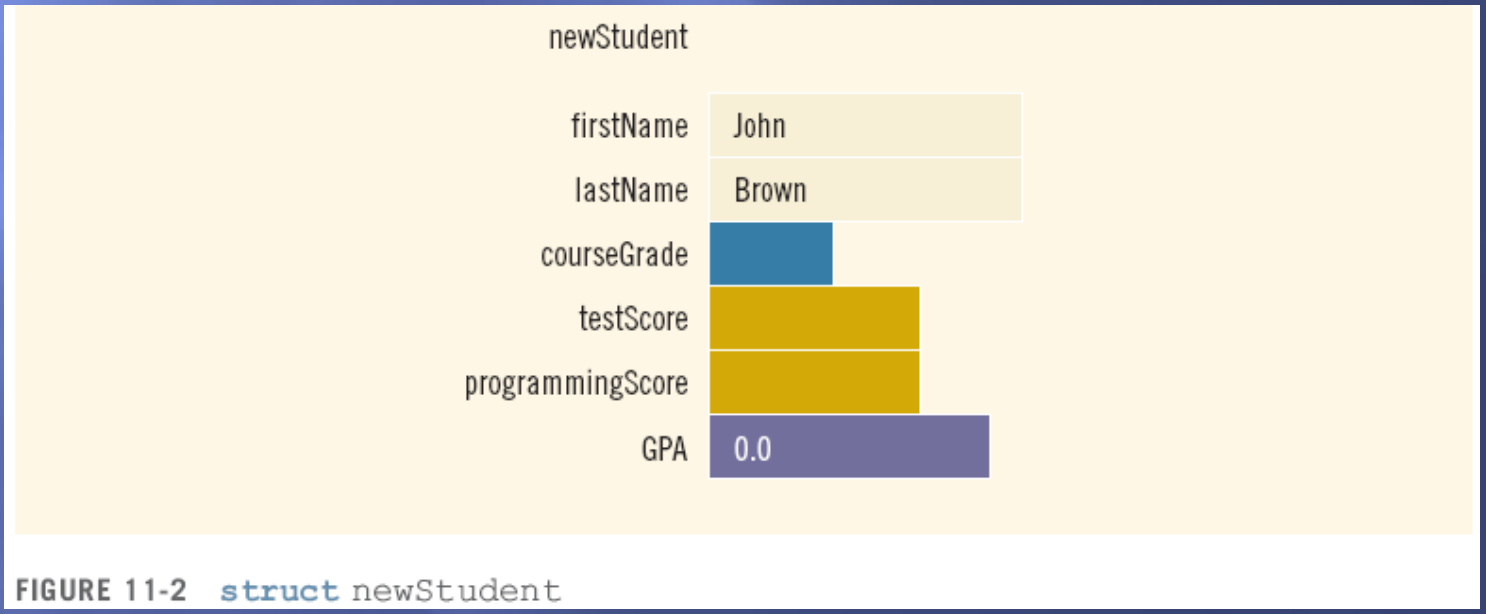

FIGURE 11-2 `struct` newStudent

Accessing struct Members (continued)

- More examples:

```
cin >> newStudent.firstName;
cin >> newStudent.testScore >>
  newStudent.programmingScore;
score = (newStudent.testScore +
  newStudent.programmingScore) / 2;
if (score >= 90)
  newStudent.courseGrade = 'A';
else if (score >= 80)
  newStudent.courseGrade = 'B';
else if (score >= 70)
  newStudent.courseGrade = 'C';
else if (score >= 60)
  newStudent.courseGrade = 'D';
else
  newStudent.courseGrade = 'F';
```

Assignment

- Value of one `struct` variable can be assigned to another `struct` variable of the same type using an assignment statement
- The statement:

```
student = newStudent;
```

copies the contents of `newStudent` into `student`

Assignment (continued)

- The assignment statement (Deep Copy):

```
student = newStudent;
```

is equivalent to the following statements:

```
student.firstName = newStudent.firstName;
student.lastName = newStudent.lastName;
student.courseGrade = newStudent.courseGrade;
student.testScore = newStudent.testScore;
student.programmingScore =
                    newStudent.programmingScore;
student.GPA = newStudent.GPA;
```

Comparison (Relational Operators)

- Compare `struct` variables member-wise
  - No aggregate relational operations allowed
- To compare the values of `student` and `newStudent`:

```
if (student.firstName == newStudent.firstName &&
    student.lastName == newStudent.lastName)
.
.
.
```

Input/Output

- No aggregate input/output operations on a `struct` variable
- Data in a `struct` variable must be read one member at a time
- The contents of a `struct` variable must be written one member at a time

```
cout << newStudent.firstName << " " << newStudent.lastName
     << " " << newStudent.courseGrade
     << " " << newStudent.testScore
     << " " << newStudent.programmingScore
     << " " << newStudent.GPA << endl;
```

struct Variables and Functions

- A struct variable can be passed as a parameter by value or by reference

```
void printStudent(studentType student)
{
    cout << student.firstName << " " << student.lastName
         << " " << student.courseGrade
         << " " << student.testScore
         << " " << student.programmingScore
         << " " << student.GPA << endl;
}
```

- A function can return a value of type struct

Arrays versus structs

TABLE 11-1 Arrays vs. structs

Aggregate Operation	Array	struct
Arithmetic	No	No
Assignment	No	Yes
Input/output	No (except strings)	No
Comparison	No	No
Parameter passing	By reference only	By value or by reference
Function returning a value	No	Yes

Arrays in structs

- Two key items are associated with a list:
  - Values (elements)
  - Length of the list
- Define a `struct` containing both items:

```
const int ARRAY_SIZE = 1000;

struct listType
{
    int listElem[ARRAY_SIZE];     //array containing the list
    int listLength;               //length of the list
};
```

Arrays in structs (continued)

```
const int ARRAY_SIZE = 1000;

struct listType
{
    int listElem[ARRAY_SIZE];     //array containing the list
    int listLength;               //length of the list
};
```

```
listType intList;
```

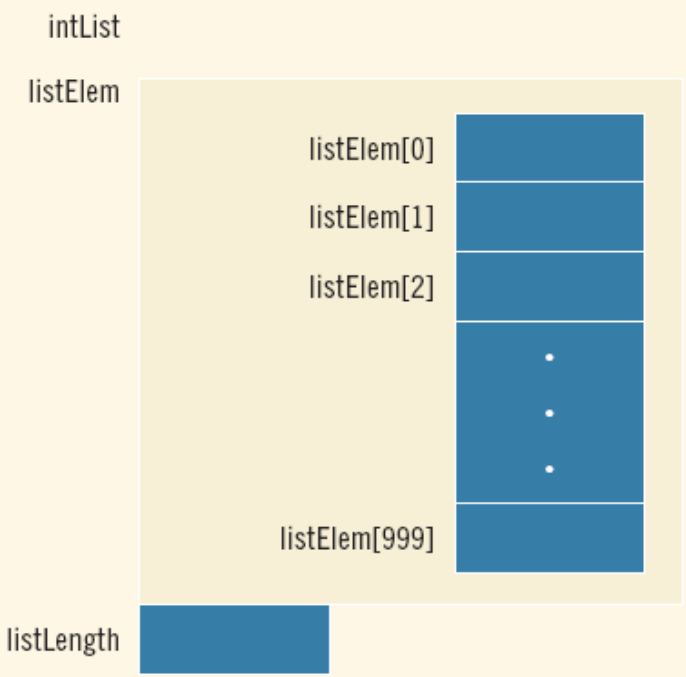

FIGURE 11-5 `struct` variable `intList`

```
intList.listLength = 0;          //Line 1
intList.listElem[0] = 12;        //Line 2
intList.listLength++;            //Line 3
intList.listElem[1] = 37;        //Line 4
intList.listLength++;            //Line 5
```

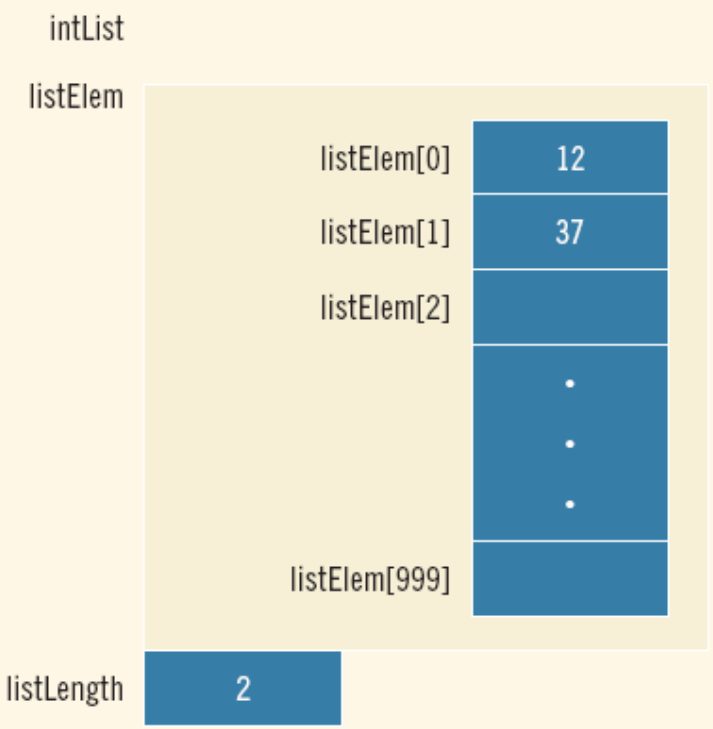

FIGURE 11-6 `intList` after the statements in Lines 1 through 5 execute

structs in Arrays

```
struct employeeType
{
    string firstName;
    string lastName;
    int    personID;
    string deptID;
    double yearlySalary;
    double monthlySalary
    double yearToDatePaid;
    double monthlyBonus;
};
```

```
employeeType employees[50];
```

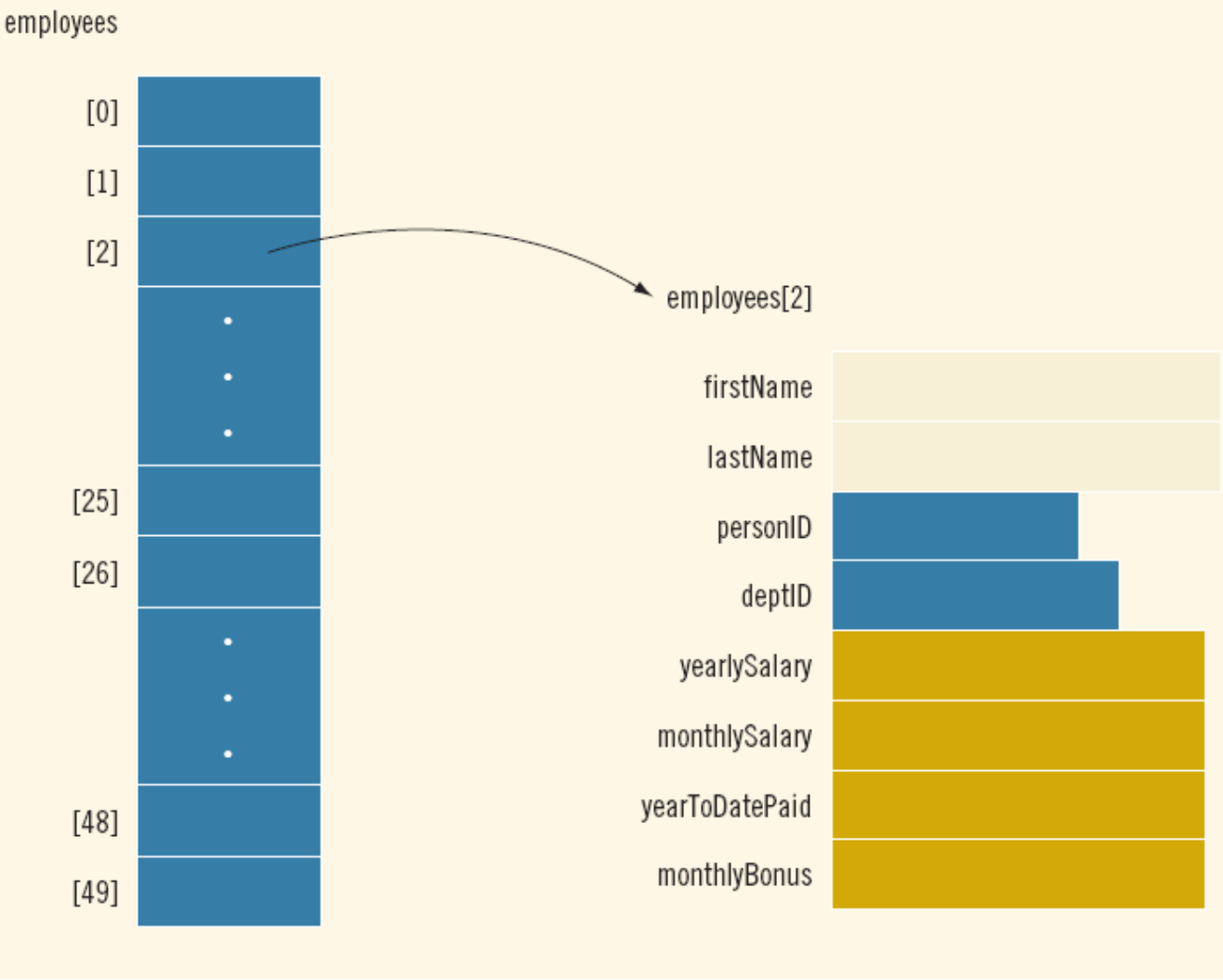

FIGURE 11-7 Array of `employees`

```
ifstream infile; //input stream variable
                 //assume that the file employee.dat has been opened
for (counter = 0; counter < 50; counter++)
{
    infile >> employees[counter].firstName
           >> employees[counter].lastName
           >> employees[counter].personID
           >> employees[counter].deptID
           >> employees[counter].yearlySalary;
    employees[counter].monthlySalary =
                  employees[counter].yearlySalary / 12;
    employees[counter].yearToDatePaid = 0.0;
    employees[counter].monthlyBonus = 0.0;
}
```

```
double payCheck; //variable to calculate the paycheck

for (counter = 0; counter < 50; counter++)
{
    cout << employees[counter].firstName << " "
         << employees[counter].lastName << " ";

    payCheck = employees[counter].monthlySalary +
               employees[counter].monthlyBonus;

    employees[counter].yearToDatePaid =
                          employees[counter].yearToDatePaid +
                          payCheck;

    cout << setprecision(2) << payCheck << endl;
}
```

structs within a struct

```
struct  employeeType
{
    string firstname;
    string middlename
    string lastname;
    string empID;
    string address1;
    string address2;
    string city;
    string state;
    string zip;
    int hiremonth;
    int hireday;
    int hireyear;
    int quitmonth;
    int quitday;
    int quityear;
    string phone;
    string cellphone;
    string fax;
    string pager;
    string email;
    string deptID;
    double salary;
};
```

versus

```
struct addressType
{
    string address1;
    string address2;
    string city;
    string state;
    string zip;
};

struct dateType
{
    int month;
    int day;
    int year;
};

struct contactType
{
    string phone;
    string cellphone;
    string fax;
    string pager;
    string email;
};
```

```
struct nameType
{
    string first;
    string middle;
    string last;
};
```

```
struct employeeType
{
    nameType name;
    string empID;
    addressType address;
    dateType hireDate;
    dateType quitDate;
    contactType contact;
    string deptID;
    double salary;
};
```

```
employeeType newEmployee;
```

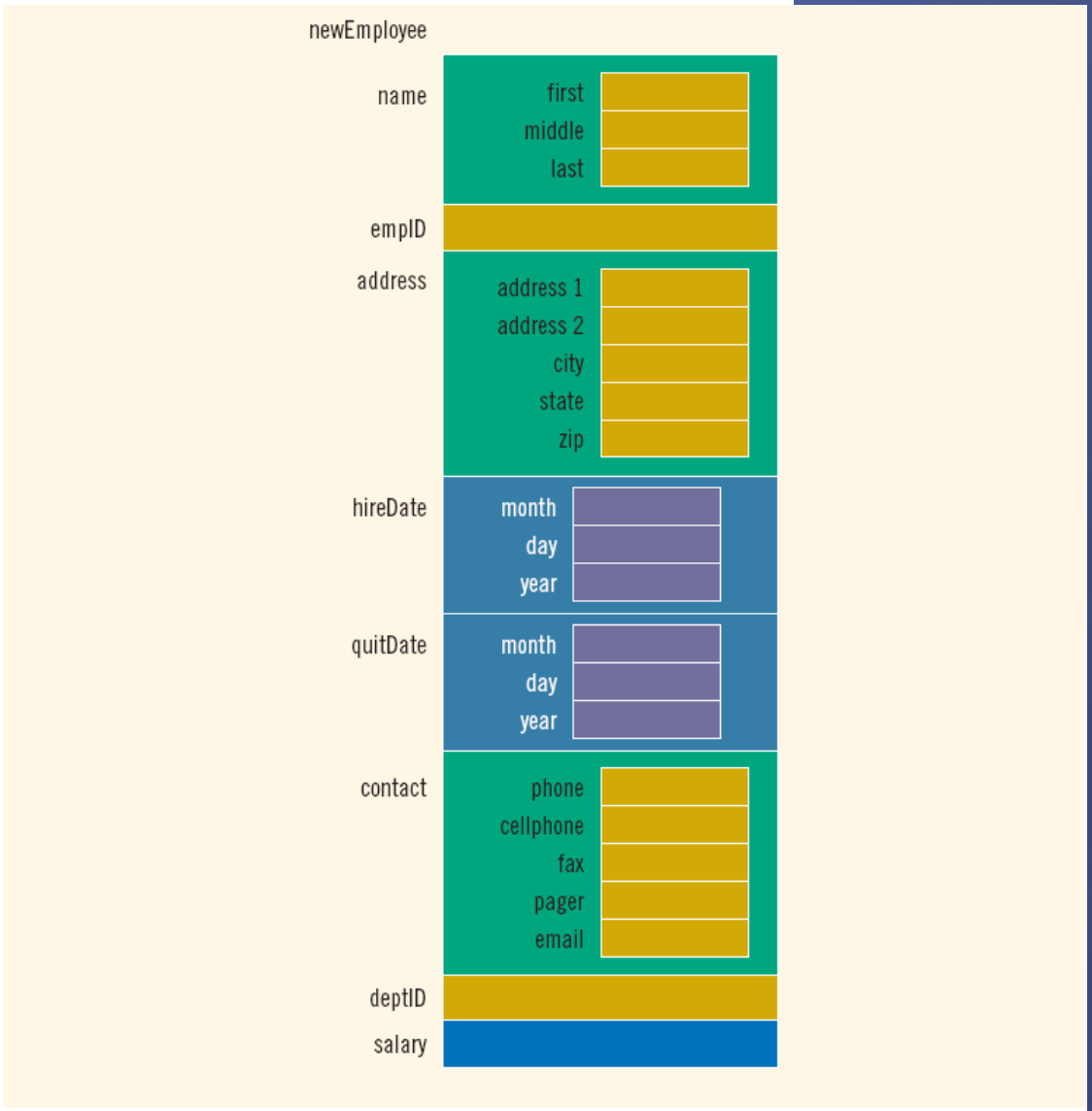

FIGURE 11-8 `struct` variable `newEmployee`

Pointer to a struct

```
struct PatientRec
{
    int ID;
    int Height;
    int Weight;
};


int main(int argc, const char * argv[]) {
    PatientRec Patient;
    PatientRec *PatientPtr = new PatientRec;


    PatientPtr->ID = 2345;
    PatientPtr->Height=160;
    PatientPtr->Weight=70;

    cout<<"Struct using pointers:"<<endl;
    cout<<PatientPtr->ID<<'\t'<<PatientPtr->Height<<'\t'<<PatientPtr->Weight<<endl;

    cout<<"Normal Struct:"<<endl;
    cout<<Patient.ID<<'\t'<<Patient.Height<<'\t'<<Patient.Weight<<endl;

    return 0;
}
```

```
Struct using pointers:
2345    160 70
Normal Struct:
1234    50  120
```

Pointer to a struct

```
struct PatientRec
{
    int ID;
    int Height;
    int Weight;
};


int main(int argc, const char * argv[]) {
    PatientRec Patient;

    PatientRec* PatientPtr = & Patient;

    PatientPtr->ID = 2345;
    PatientPtr->Height=160;
    PatientPtr->Weight=70;

    cout<<"Struct using pointers:"<<endl;
    cout<<PatientPtr->ID<<'\t'<<PatientPtr->Height<<'\t'<<PatientPtr->Weight<<endl;

    cout<<"Normal Struct:"<<endl;
    cout<<Patient.ID<<'\t'<<Patient.Height<<'\t'<<Patient.Weight<<endl;

    return 0;
}
```

```
Struct using pointers:
2345    160 70
Normal Struct:
2345    160 70
```

Files with a struct

File Open in text and binary files modes

```
ofstream Out, OutBin;
ifstream In, InBin;
int No;
Out.open("StudentsData.txt");
OutBin.open("StudentsData.bin", ios::binary);
cout<<"Enter the number of students:\t";
cin>>No;
Student* StdRec = new Student [No];
```

```
In.open("StudentsData.txt");
InBin.open("StudentsData.bin",ios::binary);
```

Files with a struct

Reading and writing from text and binary files

```
void ReadStudentRec (ifstream& InFile, Student &Std){
    if(InFile.good())
        InFile>>Std.FirstName>>Std.LastName>>Std.GPA>>Std.Fifth>>Std.Mid>>Std.Final>>Std.CourseWork;
    else {cout<<"error in Input file"<<endl; exit(0);}
}
//-------------------------------------------------------------------------
void ReadBinStudentRec (ifstream& InFile, Student &Std){
    if(InFile.good())
        InFile.read((char *) &Std, sizeof(Std));
    else {cout<<"error in Input file"<<endl; exit(0);}
}
```

```
//-------------------------------------------------------------------------
void WriteStudentRec (ofstream& OutFile, Student Std){
    OutFile<<Std.FirstName<<' '<<Std.LastName<<' '<<Std.GPA<<' '<<Std.Fifth<<' '
    <<Std.Mid<<' '<<Std.Final<<' '<<Std.CourseWork<<endl;
}
//-------------------------------------------------------------------------
void WriteBinStudentRec (ofstream& OutFile, Student Std){
    OutFile.write((char *) &Std, sizeof(Std));
}
//
```