Structures
Lecture

# OOP/Computer Programming

Instructor: Dr. Danish Shehzad

# Structures

- A *Structure* is a user's implementation of a data abstraction
- we can define a data structure to describe a group of related data, such as a "record" in a file.
- Structure is a collection of items of different data types

e.g.

| ID Number | Family Name | Given Names | Date of Birth |
|---|---|---|---|

*Student record (definition)*

| 11112222 | "Citizen" | "John Andrew" | "12/04/1989" |
|---|---|---|---|

Example (content of such a record)

# Declaring Data Structures in C++

Syntax :-

```
struct <structName>
{
        <type> <memberName1>;
        <type> <memberName2>;
        <type> <memberName3>;
        ......
};
```

# Example: Declaring a C++ struct

```
struct Date
{
    int day;
    int month;
    int year;
};
```

*structure* `name`

`members` *of the structure*

*(sometimes called "fields")*

This merely *declares a* ***new data type*** called Date.
You can then use it to create variables of type Date.

Important:- Date is not a variable. There is no memory allocated for it. It is merely a ***type*** (like int, float, etc).

# Defining a Structure Variable

## Syntax :-

```
<structName> <variableName>;
```

## Examples:

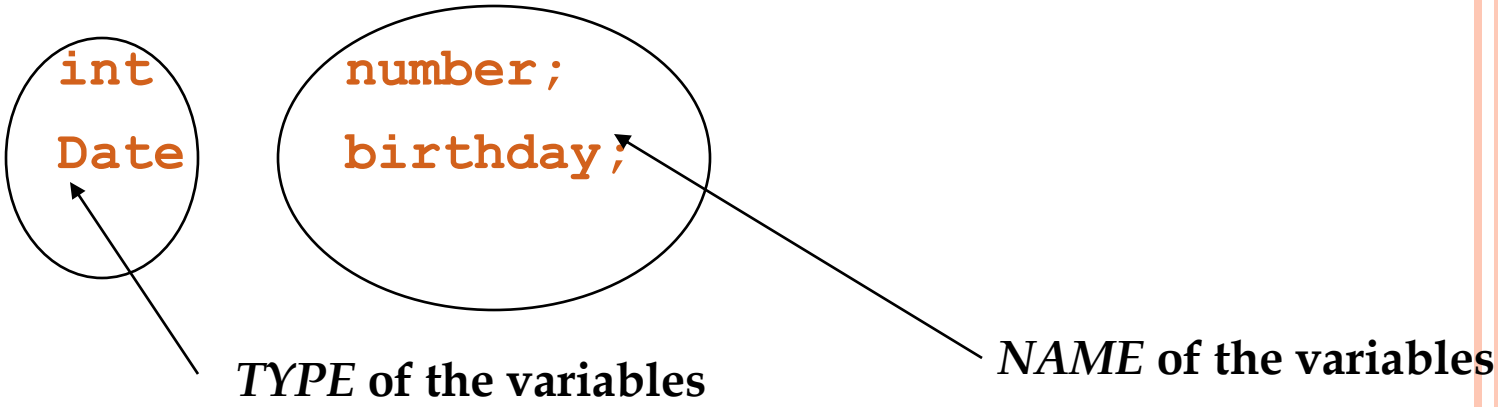
```
Date birthday;
```

- creates a variable called **birthday** of type **Date**. This variable has 3 *components* (members) : **day**, **month**, and **year**.

```
Date today;
```

- creates another variable of type **Date**, also with component parts called **day**, **month** and **year**.

# Defining a Structure Variable Vs Defining a "normal" Variable




note the consistent format :

```
<type> <variableName>;
```

# Initializing Structure Type Variables

```
struct Name
{
   char first[30];
   Char last[30];
};

Name poet_name;
strcpy(poet_name.first,”Allama);
strcpy(poet_name.last,”Iqbal”);
```

Note :
Values of the members need to be Assigned individually, AFTER the variable is created.

| | |
|---|---|
| “Allama" | “Iqbal" |

# Members of Different Types

```
struct Student
{
    ... id;
    ... name;
    ... age;
    ... gender;
};
student std;
std.id = 1234;
strcpy(std.name,"Hassan Ali");
std.age = 19;
std.gender = 'M';
```

*The members of a* struct *need not be of the same type.*

*What should be the types of these members?*

| | | | | |
|---|---|---|---|---|
| ali | 1234 | "Hassan Ali" | 19 | 'M' |

# Creating structure of Library Database

| ISBN | Book Name | Author Name | Publisher | Number of Copies | Year of Publish |
|---|---|---|---|---|---|
| 1293 | Network Security | Martin | Waley | 4 | 1998 |
| 9382 | Data mining | Muhammad Zaki | Wrox | 6 | 2003 |
| 9993 | Data warehousing | Stephen Brobst | MIT | 8 | 2003 |
| 3423 | C Programming | M. Kamber | Waley | 4 | 1996 |

```
struct Library
{
    int ISBN, copies, PYear;
    char bookName[30], AuthorName[30], PublisherName[30];
};
```

# Accessing Structure Members

```
Library libraryVariable;
cin >> libraryVariable.ISBN;
cin >> libraryVariable.bookName;
cin >> libraryVariable.AuthorName;

cout << libraryVariable.ISBN <<
libraryVariable.bookName << libraryVariable.AuthorName;

int tempISBN = libraryVariable.ISBN + 1;
```

*The dot is called the "member" operator*

# Accessing Structures

```
cout << bookName;
```

Error! // bookName is not a variable. It is only the name of a member in a structure

```
cout << Library.bookName;
```

Error! // Library is not the name of a variable. It is the name of a type

# Common Errors in Accessing Structures (contd.)

```
cout << libraryVariable;
```

```
//cout does not know how to handle the variable
libraryVariable, as it is not one of the built-in types.
You have to give it individual bits of libraryVariable
that it can recognize and handle.
```

```
cout << libraryVariable.ISBN << libraryVariable.bookName;
//this is OK
```

# Accessing Structure Variables (Example 1)

```
void main (void)
{
    struct Library
    {
        int ISBN, copies, PYear;
        char bookName[30], AuthorName[30], PublisherName[30];
    };
    Library libraryVariable;

    libraryVariable.ISBN = 1293;
    strcpy (libraryVariable.bookName, "Network Security");
    strcpy (libraryVariable.AuthorName, "Martin");
    strcpy (libraryVariable.PublisherName, "Waley");
    libraryVariable.copies = 4;
    libraryVariable.PYear = 1998;

    cout << libraryVariable.ISBN << libraryVariable.bookName <<
    libraryVariable.AuthorName << libraryVariable.PublisherName <<
    libraryVariable.copies << libraryVariable.PYear;
}
```

# Accessing Structure Variables (Example 1)

```
void main (void)
{
    struct Library
    {
        int ISBN, copies, PYear;
        char bookName[30], AuthorName[30], PublisherName[30];
    };
    Library libraryVariable1, libraryVariable2, libraryVariable3,
    libraryVariable4;

    Library LibraryArray [4]; // alternative and easiest way
}
```

# Assignment to Structure Variable

- The value of a structure variable can be assigned to another structure variable ***of the same type***, e.g :

```
Library libraryVariable1, libraryVariable2;
strcpy (libraryVariable1.bookName , "C Programming");
libraryVariable1.ISBN = 1293;
libraryVariable2 = libraryVariable1;
cout << libraryVariable2.bookName << libraryVariable2.ISBN;
```

- Assignment is the only operation permitted on a structure. **We can not add, subtract, multiply or divide structures**.

# Structures within Structures

```
void main ()
{
  struct University
  {          char Name [30];
             char city [30];
             Library libraryVariable;
  };
  University universityVariable;
  strcpy (universityVariable.Name, “NU-FAST”);
  strcpy (universityVariable.city, “Islamabad”);
  universityVariable.libraryVariable.ISBN = 1293;
  strcpy (universityVariable.libraryVariable.bookName,
  “C programming”);
}
```

# Accessing Structure in Structure

```
cin >>
  universityVariable.libraryVariable.bookName;
cin >>
  universityVariable.libraryVariable.ISBN;


cout <<
  universityVariable.libraryVariable.bookName
  << universityVariable.libraryVariable.ISBN;
```

# Passing Structure Variables as Parameters

- *An individual structure member* may be passed as a *parameter to a function*, e.g. :
  - validLibraryData (libraryVariable.ISBN);

- *An entire structure variable may be passed* , e.g. :
  - validLibraryData (libraryVariable);

# Example : Passing a Structure Member

```
void validLibraryData (int ISBN);
void main(void)
{
  //assuming that Library structure has
  already defined
  Library libraryVarialbe;
  validLibraryData (libraryVariable.ISBN);
}
void validLibraryData (int ISBN)
{
  cout << “Library ISBN = “ << ISBN;
}
```

# Example : Passing an entire Structure

```
struct Library
{
        int ISBN, copies, PYear;
        char bookName[30], AuthorName[30], PublisherName[30];
};
void validLibraryData (Library var1);
void main (void)
{
   Library libraryVariable;
   libraryVariable.ISBN = 1293;
   strcpy (libraryVariable.bookName, "Network Security");
   strcpy (libraryVariable.AuthorName, "Martin");
   validLibraryData (libraryVariable);
}
void validLibraryData (Library var1)
{
   cout << "ISBN = " << var1.ISBN << "\n";
   cout << "Book name = " << var1.bookName << "\n";
}
```

# Returning a Structure Variable

```
struct Library
{
        int ISBN, copies, PYear;
        char bookName[30], AuthorName[30], PublisherName[30];
};
Library inputBookInformation (void);
void main (void)
{
   Library libraryVariable1;
   libraryVariable1 = inputBookInformation ( );
   cout << libraryVariable1.ISBN << libraryVariable1.bookName;
}
Library inputBookInformation (void)
{
    Library var1;
    var1.ISBN = 1293;
    strcpy (var1.bookName, "Network Security");
    strcpy (var1.AuthorName, "Martin");
    return var1;
}
```

# Pointers to structure variables

- *Pointers of structure variables can be declared like pointers to any basic data type*

```
Library var1, *ptrToLibrary;
ptrToLibrary = &var1;
```

- *Members of a pointer structure type variable can be accessed using (->) operator*

```
ptrToLibrary->ISBN =20;
strcpy( ptrToLibrary->bookName, “C Programming”);
```

# Pointers to structure variables (Example 1)

```
void main (void)
{
    Library libraryVariable1, *PtrToLibrary;
    libraryVariable.ISBN = 1293;
    strcpy (libraryVariable.bookName, “Network Security”);
    strcpy (libraryVariable.AuthorName, “Martin”);
    strcpy (libraryVariable.PublisherName, “Waley”);
    libraryVariable.copies = 4;
    libraryVariable.PYear = 1998;

    PtrToLibrary = &libraryVariable1;
    PtrToLibrary->ISBN = 3923;
    PtrToLibrary->copies = 10;
    cout << “The values are “ << libraryVariable1.ISBN << “ , ”
          << PtrToLibrary->ISBN;
}
```

Output: The values are 3923 , 3923

# Pass by Reference structure variables to Functions (Example 1)

```
void Function1 (Library *ptr);
void main (void)
{
    Library var1;
    Function1 (&var1);
    cout << var1.ISBN << var1.bookName << var1.AuthorName;
}
void Function1 (Library *libraryVariable)
{
    libraryVariable->SBN = 1293;
    strcpy (libraryVariable->bookName, "Network Security");
    strcpy (libraryVariable->AuthorName, "Martin");
    strcpy (libraryVariable->PublisherName, "Waley");
    libraryVariable->copies = 4;
    libraryVariable->PYear = 1998;
```

Output: 1293 Network Security Martin

# Array of Structure (Example 1)

```
void main (void)
{
    Library libraryArray [4];
    libraryArray[0].ISBN = 1293;
    strcpy (libraryArray[0].bookName , "Network Security");

    libraryArray[1].ISBN = 9832;
    strcpy (libraryArray[1].bookName, "C Programming");

    libraryArray[2].ISBN = 3832;
    strcpy (libraryArray[2].bookName , "Technical Report Writing");

    cout << libraryArray[0].ISBN << libraryArray[1].ISBN <<
        libraryArray[2].ISBN;

    cout << libraryArray[0].bookName << libraryArray[1].bookName <<
        libraryArray[2].bookName;
}
```

# Dynamic Memory Allocation (DMA) of Structure Type Variables

- *We can also dynamically allocate the memory of **any structure type** variable using new operator.*
- *For example.*
  - `Library *PtrToLibrary;`
  - `PtrToLibrary = new Library;`

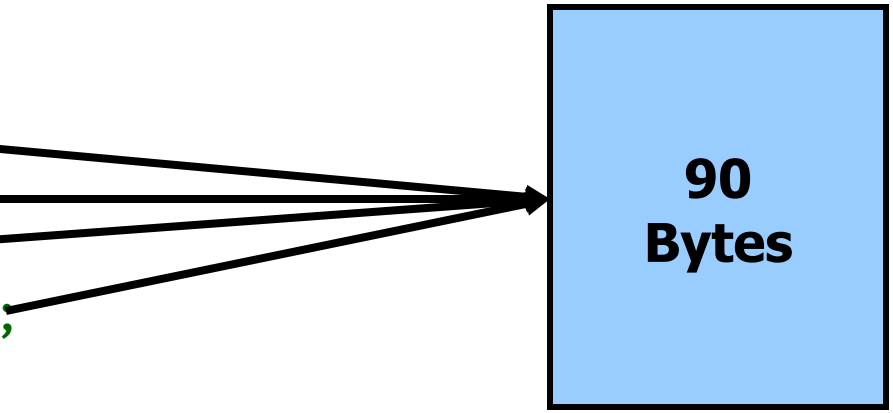
- Very similar to
  - `float *PtrTofloat;`
  - `PtrTofloat = new float;`
- We can delete memory allocated at execution time using ***delete***
  - `delete PtrToLibrary;`

# Union

- As structures, unions are also used to group a number of **different variables together**.
- The difference between union and structure is that, structure treat each of its member as a **different memory location** store in the main memory.
- While union treat each of its **member as a single memory location** store in the main memory.
  - i.e. all of the members of union shares a common memory of union member.

# Union Example

```
union searchOption
{
        int SearchByRollNumber;
        char SearchByName[90];
        char SearchByAddress[90];
        char SearchByPhoneNumber[90];
};
searchOption sv;
void main (void)
{
        int option = 0;
        switch (option)
        {
          case 0:  FunSearchRoll (sv.SearchByRollNumber); break;
          case 1:  FunSearchName(sv.SearchByName); break;
          case 2:  FunSearchByAddress(sv.SearchByAddress); break;
          case 3:  FunSearchByPhone(sv.SearchByPhoneNumber);
           break;
        }
}
```



# Some examples

# Union Example

```
union foo {
  int a;   // can't use both a and b at once
  char b;
} foo;

struct bar {
  int a;   // can use both a and b simultaneously
  char b;
} bar;

union foo x;
x.a = 3; // OK
x.b = 'c'; // NO! this affects the value of x.a!

struct bar y;
y.a = 3; // OK
y.b = 'c'; // OK
```

# When to Use UNION?

We can use the unions in the following locations.

- Share a single memory location for a variable and use the same location for another variable of different data type.
- Use it if you want to use, for example, a long variable as two short type variables.
- We don't know what type of data is to be passed to a function, and you pass union which contains all the possible data types.

```
// structs/time-1.cpp - Shows simple use of a struct.
// Fred Swartz - 2003-09-09

//=================================================== includes
#include <iostream>
using namespace std;

//=========================================== define new types
struct Time {
    int hours;
    int minutes;
    int seconds;
};


//================================================= prototypes
int toSeconds(Time now);


//===================================================== main
int main() {
    Time t;
    while (cin >> t.hours >> t.minutes >> t.seconds) {
        cout << "Total seconds: " << toSeconds(t) << endl;
    }
    return 0;
}

//================================================= toSeconds
int toSeconds(Time now) {
    return 3600*now.hours + 60*now.minutes + now.seconds;
}
```

```
#include <iostream>

using namespace std;

struct xampl {
  int x;
};

int main()
{
  xampl structure;
  xampl *ptr;

  structure.x = 12;
  ptr = &structure; // Yes, you need the & when dealing with structures
                    //  and using pointers to them
  cout<< ptr->x;    // The -> acts somewhat like the * when used with pointers
                    //  It says, get whatever is at that memory address
                    //  Not "get what that memory address is"
  cin.get();
}
```

```
struct GradeRec
{
  float percent;
  char grade;
};
struct StudentRec
{
  string lastName;
  string firstName;
  int age;
  GradeRec courseGrade;
};
void main(void)
{
  StudentRec student;
  cout << "Enter first name: ";  cin >> student.firstName;
  cout << "Enter last name: ";   cin >> student.lastName;
  cout << "Enter age: ";            cin >> student.age;

  cout << "Enter overall percent: ";
  cin >> student.courseGrade.percent;
```

```
if(student.courseGrade.percent >= 90)
  {
     student.courseGrade.grade = 'A';
  }
  else if(student.courseGrade.percent >= 75)
  {
     student.courseGrade.grade = 'B';
  }
else {
     student.courseGrade.grade = 'F';
  }
  cout << "\n\nHello " << student.firstName << ' ' << student.lastName
       << ".  How are you?\n";
  cout << "\nCongratulations on reaching the age of " << student.age
       << ".\n";
  cout << "Your overall percent score is "
       << student.courseGrade.percent << " for a grade of "
       << student.courseGrade.grade;
}
```

OUTPUT:
Enter first name: Sally
Enter last name: Smart

Enter age: 19 ....

```
struct PersonRec
{
  string lastName;
  string firstName;
  int age;
};
typedef PersonRec PeopleArrayType[10]; //an array of 10 structs
void main(void)
{
PeopleArrayType people; //a variable of the array type
  for (int i = 0; i < 10; i++)
  {
     cout << "Enter first name: ";
     cin >> people[i].firstName;
     cout << "Enter last name: ";
     cin >> people[i].lastName;
     cout << "Enter age: ";
     cin >> people[i].age;
  }
  for (int i = 0; i < 10; i++)
  {
     cout << people[i].firstName << ' ' << people[i].lastName
          << setw(10) << people[i].age;

} }
```

- Questions?