

Goals for This Lecture:

- Briefly Introduce the Idea of Structures and Classes in C++ and F95
- Understand the Object-Oriented Programming idea of Encapsulation

C++ Structures

- C++ and F95 have the ability to allow you to create your own data types that can contain multiples components
- In C++ such entities are called structures
- In F95 such entities are called derived types
- Lets first look at how structures are created in C++
- Example:

```
#include<iostream>
using namespace std;
struct Person{           // Define the structure
    int birthMonth;      // Define the member variables
    int birthDay;        // a.k.a. "components"
    int birthYear;
    char lastName[];
    char firstName[];
};
Person albert;          // Declare some instances of Person
Person enrico;
```

More on C++ structures

- C++ structures can be initialized in the declaration
- Example:

```
Person albert = {3 14 1879 "Einstein" "Albert"};
```

- Another way in which the components of C++ structures can be initialized and referenced is by using the "." (dot) notation:

Example:

```
Person albert;
```

```
albert.birthMonth = 3;
```

```
albert.birthDay = 14;
```

```
albert.birthYear = 1879;
```

```
strcpy(albert.lastName, "Einstein");
```

```
strcpy(albert.firstName, "Albert");
```

F95 Derived Types

- In F95 the equivalent of a C++ structure is called a derived type
- Let's look at an example:

```
type person          !Define the structure
  integer :: birth_month      ! Define the components
  integer :: birth_day
  integer :: birth_year
  character(len=30) :: last_name
  character(len=30) :: first_name
end type person

type(person) :: albert      // Declare some instance
type(person) :: enrico     // of the derived-type person
```

More on F95 Derived Types

- In F95 a derived type can be initialized by a *structure constructor*
- Example:

```
type(person) :: albert=person(3, 14,1879,"Albert","Einstein")
```

- In F95 the components of a derived type can be initialized and referenced using the “%” notation
- Example:

```
albert%birth_month=3  
albert%birth_day=14  
albert%birth_year=1879  
albert%first_name="Albert"  
albert%last_name="Einstein"
```

Arrays of structures or derived types

- It is possible to create arrays of structures in C++ or derived types in F95
- C++ example:

```
Person students[25];
students[0].birthMonth=3 ;
students[0].birthDay=14 ;
students[0].birthYear=1879 ;
strcpy(students[0].firstName,"Albert") ;
strcpy(students[0].lastName,"Einstein") ;
```

- F95 example:

```
type(person) :: students(25)
students(1)%birth_month=3
students(1)%birth_day=14
students(1)%birth_year=1879
students(1)%first_name="Albert"
students(1)%last_name="Einstein"
```

An example: Reading a database pt. 1

```
program read_database
implicit none
integer, parameter :: nstudents=25
type person      !Define the structure
  integer :: birth_month      ! Define the components
  integer :: birth_day
  integer :: birth_year
  character(len=30) :: last_name
  character(len=30) :: first_name
end type person

type(person) :: students(nstudents)
integer :: i, ierr
integer, parameter :: lun=17
```

An example: Reading a database pt. 2

```
                                ! Open the database file
open(unit=lun1,file='students.dat',status='old',iostat=ierr)
if(ierr /= 0) then
    write(*,*) ' Read error'
    stop
endif

do i=1,nstudents,1      ! Read in values for each student
    read(lun1,*,iostat=ierr) students(i)%last_name,    &
                                students(i)%first_name, &
                                students(i)%birth_month, &
                                students(i)%birth_day,  &
                                students(i)%birth_year

enddo

close(unit=lun1)      ! Close the file
stop
end program read_database
```

C++ Classes

- A C++ class is similar to a structure but can contain member functions as well as member variables
- Example:

```
#include<iostream>
using namespace std;
class Person{          // Define the structure
public:
    void printBirthDate(); // Declare a member function
    int birthMonth;       // Define the member variables
    int birthDay;        // a.k.a. "components"
    int birthYear;
    char lastName[];
    char firstName[];
};
void Person::printBirthDate() { // Define the member function
cout << birthMonth << "/" << birthDay
      << "/" << birthYear << endl;

int main() {
Person albert={3,14,1879,"Einstein","Albert"};
albert.printBirthDate();
return(0); }
```

More on C++ Classes

- The **public:** directive declares that the items following the directive are accessible to anyone
 - A **private:** directive would declare that anything following the directive is accessible only to member functions
- The void function **printBirthDate()** is a *member function*, also known as a *method*, of the class
- **albert** is said to be an *instance* of the **Person** class
- The **printBirthDate()** method is invoked for the class instance **albert** by means of the *dot operator*: **albert.printBirthDate()**
- The definition of the method **printBirthDate()** is identified as being a member of the class **Person** by means of the *scope operator* “**::**” which is used to tell the compiler that the function **Person::printBirthDate()** has a scope that is restricted to the class **Person**

Encapsulation

- The features of classes can be used to restrict or hide access to certain data
- This prevents users of the classes from inadvertently altering a critical piece of data or gaining access to a sensitive data item.
- This programming strategy is called *encapsulation* or *data hiding*
- Critical data items are declared as private
 - These items are accessed only through member functions
- Encapsulation is an important cornerstone of *object-oriented programming (OOP)*
- Encapsulation promotes re-use of code by exposing only the minimum amount of data to the user of a class

An Example of Encapsulation

- Let's use the Person class to illustrate how encapsulation works
- We'll add an ID number member variable to the class that must remain protected at all costs
 - We'll define a member function called **setID** that requires a **passcode** to set the ID number
 - We'll define a **compareID** function that allows users to compare a number to the user's ID number
- We'll put the class definition in a header file that the users can access
- We'll put the member functions (methods) in a separate file that can be compiled by itself
- Only the **Person_methods.o** file will be made available to the general users
 - The passcode remains inaccessible to the general user
 - **Note: this method is not perfect and has security flaws**

Encapsulation with C++ Classes

- First define a header file with the class definition
- Person.h:

```
#include<iostream>
using namespace std;
class Person{          // Define the structure
public:
    void printBirthDate(); // Declare a member function
    void setID(int id_arg, int pcode); // Declare setID function
    bool compareID(int id_arg); // Declare compareID function
    int birthMonth;        // Define the member variables
    int birthDay;         // a.k.a. "components"
    int birthYear;
    char lastName[];
    char firstName[];
private:
    int idNum ;          // Define private ID number
};
```

Encapsulation with C++ Classes

- First define a file, Person_methods.cpp, with the methods in it:

```
#include<iostream>
#include "Person.h"          // Access the class definition
using namespace std;
void Person::setID(int id_arg, int pcode) {          // SetID method
    if(pcode == 1234) {
        idNum = id_arg;
        cout << " ID number successfully set\n";
    } else{
        cout << " Wrong pass code dummy!\n";
        cout << pcode << endl; }
}
void Person::printBirthDate() {                    // printBirthDate method
    cout << firstName << " " << lastName << "'s Birth Date is " <<
        birthMonth << "/" << birthDay << "/" << birthYear << endl;
}
bool Person::compareID(int id_arg) {              // compareID method
    if(id_arg == idNum) return(true);
    else return(false);
}
```

Encapsulation with C++ Classes

- Access the class by including the header file

```
#include <iostream>
#include <cstring>
#include "Person.h"
using namespace std;
int main() {
    Person albert ;      // Declare an instance of Person
    albert.birthMonth=3; // Initialize info for Albert
    albert.birthDay=14;
    albert.birthYear=1879;
    strcpy(albert.firstName,"Albert");
    strcpy(albert.lastName,"Einstein");
    albert.printBirthDate() ;
    albert.setID(173,0000); // First try at setting ID
    albert.setID(173,1234); // Second try at setting ID
    cout << albert.compareID(173) << endl ; // Compare ID 173
    cout << albert.compareID(174) << endl ; // Compare ID 174
    return(0);
}
```

Encapsulation in FORTRAN 95

- This same example can be implemented in FORTRAN 95
- We'll use two derived types to define the data structure of the class
 - One type, called `id` will hold the ID number as a private component
 - The second type will include the `id` type as component
- We'll place these types in a module named `person_class`
- The methods for the class will be defined as subroutines and functions that are contained within the module
- Since F95 has no equivalent of the dot operator in C++ to access functions we'll name all of our methods starting with a `person_` to identify them as method of the `person_class`
 - FORTRAN 2003 allows methods to be accessed with a `%` operator
 - We will only make the `person_class.o` and `person_class.mod` files available to the general user

Encapsulation with F95 Modules pt 1

- person_class.f90:

```
! This module defines the person_class
```

```
module person_class
  type person_id
    private
      integer :: number
    end type person_id
  type person
    integer :: birth_month
    integer :: birth_day
    integer :: birth_year
    character(len=30) :: last_name
    character(len=30) :: first_name
    type(person_id) :: id
  end type person
contains
```

Encapsulation with F95 Modules pt 2

- person_class.f90 continued:

```
! This subroutine sets an ID number for an instance  
! of class person if the user has the correct passcode
```

```
subroutine person_set_id(someone,idn,pcode)  
implicit none  
type(person) :: someone  
integer :: idn, pcode  
if(pcode == 1234) then  
    someone%id%number = idn  
    write(*,*) ' ID number set successfully!' else  
    write(*,*) ' Wrong pass code dummy!'  
endif  
return  
end subroutine person_set_id
```

Encapsulation with F95 Modules pt 3

- person_class.f90 continued:

```
! This subroutine prints out the birthday of
! An instance of type person
subroutine person_print_birth_date(someone)
implicit none
type(person) :: someone
write(*,*) someone%first_name," ", &
someone%last_name,      &
"'s birthday is ", &
someone%birth_day,"/", &
someone%birth_month,"/", &
someone%birth_year,"/"
return
end subroutine person_print_birth_date
```

Encapsulation with F95 Modules pt 4

- person_class.f90 continued:

```
! This function compares an ID number and returns  
! .true. or .false.
```

```
function person_compare_id(someone,idn) result(prop)  
implicit none type(person) :: someone  
integer :: idn  
logical :: prop  
prop = (someone%id%number == idn)  
return  
end function person_compare_id
```

```
end module person_class
```

Encapsulation with F95 Modules pt 5

- encapsulation.f90 continued:

```
! This program uses the person_class and it's methods
```

```
program encapsulation
```

```
use person_class
```

```
implicit none
```

```
type(person) :: albert
```

```
albert%birth_day=14
```

```
albert%birth_month=3
```

```
albert%birth_year=1879
```

```
albert%last_name="Einstein"
```

```
albert%first_name="Albert"
```

```
call person_print_birth_date(albert)
```

```
call person_set_id(albert,173,0000)
```

```
call person_set_id(albert,173,1234)
```

```
write(*,*) person_compare_id(albert,174)
```

```
write(*,*) person_compare_id(albert,173)
```

```
stop
```

```
end program encapsulation
```

Assignment

- Read Savitch Sections 6.1 & 6.2.
- Read Chapman Sections 8.5 & 9.8