UTM
UNIVERSITI TEKNOLOGI MALAYSIA

Introduction to Abstract Data Type & C++ Revision

Nor Bahiah Hj Ahmad & Dayang Norhayati A.Jawawi
School of Computing

Objectives

At the end of the class students are expected to:

- Understand Abstract Data Type concept
- Review C++ programming
  - Declaring a class, data member and function member
  - Creating constructor and destructor
  - Pass object as function parameter
  - Return object from a function
  - Array of class
  - Pointer to class
  - Define and implement a class within header files and implementation files

Abstraction

Abstract data type (ADT)

- A collection of data and a set of operations on the data
- You can use an ADT's operations without knowing their implementations or how data is stored, if you know the operations' specifications

Abstraction

- Separates the purpose of a module from its implementation
- Specifications for each module are written before implementation

Abstraction and Information Hiding

Data abstraction

- Focuses on the operations of data, not on the implementation of the operations
- Asks you to think *what* you can do to a collection of data independently of *how* you do it
- Allows you to develop each data structure in relative isolation from the rest of the solution
- A natural extension of functional abstraction

Functional abstraction

- Separates the purpose of a module from its implementation

Abstraction and Information Hiding

Information hiding

- Hide details within a module
- Ensure that no other module can tamper with these hidden details
- Makes these details inaccessible from outside the module
- Public view of a module
  - Described by its specifications
- Private view of a module
  - Implementation details that the specifications should not describe

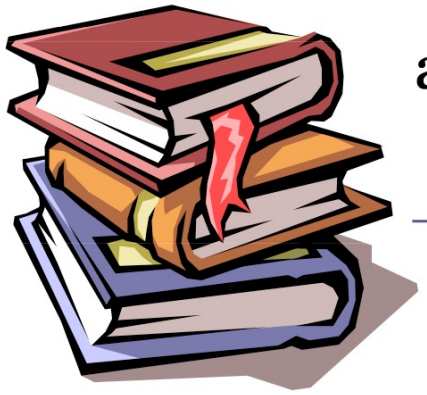
Abstract data type (ADT)

Abstract data type (ADT)

- An ADT is composed of
  - A collection of data
  - A set of operations on that data
- Specifications of an ADT indicate
  - What the ADT operations do, not how to implement them
- Implementation of an ADT
  - Includes choosing a particular data structure

Abstract data type (ADT)

Abstraction of a book



abstract to

book
title year author publisher price
getData() print() checkPrice() checkPublisher()

attributes

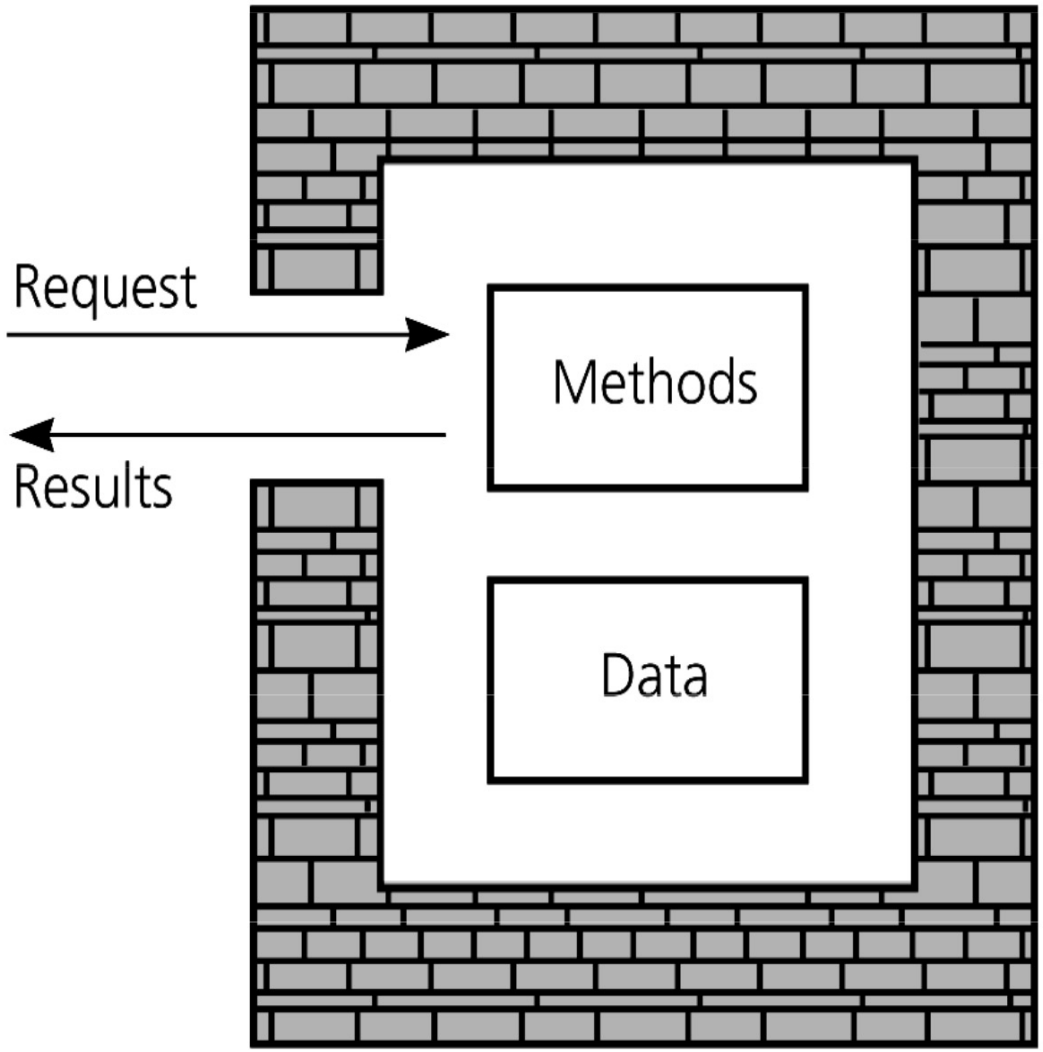
behavior

Encapsulation

- Encapsulation is the process of combining data and functions into a single unit called class.
- The programmer cannot directly access the data. Data is only accessible through the functions present inside the class.
- Data encapsulation led to the important concept of data hiding.
- Data hiding is the implementation details of a class that are hidden from the user. The concept of restricted access led programmers to write specialized functions or methods for performing the operations on hidden members of the class.

baliah@utm.my

Encapsulation



An object's data and methods are encapsulated

C++ Classes

Encapsulation combines an ADT's data with its operations to form an object

- An object is an instance of a class
- A class defines a new data type
- A class contains data members and methods (member functions)
- By default, all members in a class are private But can be specified as public

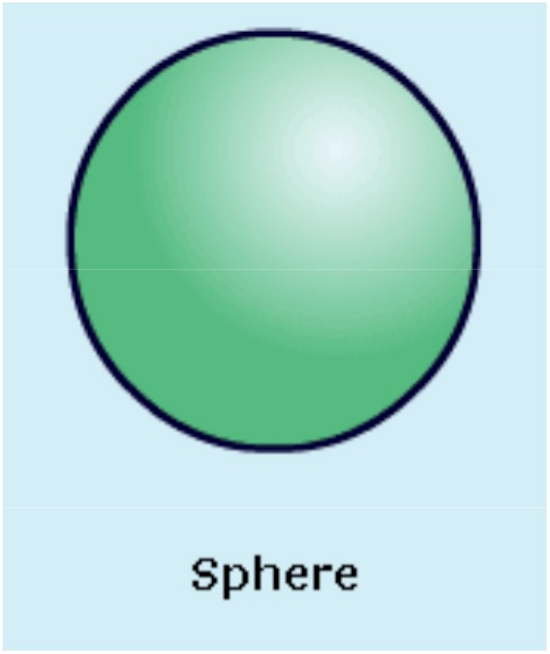
C++ Class definition

```
class clasName
{
public:
    list of data member declaration;
    list of function member declaration;
private:
    list of data member declaration;
    list of function member declaration;
};  // end class definition
```

class members: data and function

public : members that are accessible by other modules
private : members that are hidden from other modules and can only be accessed by function member of the same class.

Example : sphere




```
class Sphere
```

```
setRadius
getRadius
getDiameter
getCircumference
getArea
getVolume
```

```
Radius;
```

Class definition

```
class Sphere
{
public:
  Sphere();  // Default constructor
  Sphere(double initialRadius); // Constructor
  void setRadius(double newRadius);
  double getRadius()const; //can't change data members
  double getDiameter() const;
  double getCircumference() const;
  double getArea() const;
  double getVolume() const;
  void displayStatistics() const;
  ~Sphere()
private:
 double theRadius;   // data members should be private
};  // end Sphere
```

Class methods/member functions

- Constructor – allocates memory for an object and can initialize new instances of a class to particular values.
- Destructor – destroys an instance of a class when the object's lifetime ends.
- C++ function
- const function – function that cannot alter data member of the class

C++ Classes: Constructors

Constructors

- Create and initialize new instances of a class
- Invoked when you declare an instance of the class
- Have the same name as the class
- Have no return type, not even void

A class can have several constructors

- A default constructor has no arguments
- The compiler will generate a default constructor if you do not define any constructors

Constructor Properties

- Can have more than one constructor (*overloading*) whereby each constructor must be distinguished by the arguments.

```
Sphere();

Sphere(double initialRadius);
```

- Default constructor: **`Sphere();`**
- Can have argument:

```
Sphere(double initialRadius);
```

- Can have default argument:

```
Sphere(double initialRadius =10.00)
```

Constructor Implementation: default constructor

The implementation of a method qualifies its name with the scope resolution operator ::

- Sets data members to initial values

```
Sphere::Sphere()  :  theRadius(1.0)
{
}   // end default constructor
```

Instance declaration:

```
Sphere unitSphere;      // theRadius is set to 1.
```

Constructor Implementation: constructor with argument

```
Sphere::Sphere(double initialRadius)
{
   if (initialRadius > 0)
      theRadius = initialRadius;
   else
      theRadius = 1.0;
}  // end constructor
```

Instance declaration:

```
Sphere mySphere(5.1);
                    // set theRadius to 5.1
```

Constructor Implementation: constructor with default argument

```
Sphere::Sphere(double initialRadius = 5.00)
{
   if (initialRadius > 0)
      theRadius = initialRadius;
   else
      theRadius = 1.0;
}  // end constructor with default argumen.
```

2 methods of to declare instance of a class:

Sphere mySphere(10.1); // set theRadius to 10.1

Sphere mySphere; // set theRadius to default value 5.0

Must avoid ambiguity error

C++ Classes: Destructors

Destructor

- Destroys an instance of an object when the object's lifetime ends
- Each class has one destructor
  - For many classes, you can omit the destructor
  - The compiler will generate a destructor if you do not define one
- Example:
  ~Sphere();

Member Function Implementation

```
double Sphere::getDiameter() const
{   return 2.0 * theRadius; }
// end getDiameter
```

Method to call the member function:

- From main() or nonmember function

```
cout << mySphere.getDiameter() << endl;
```

- From member function

```
void Sphere::displayStatistics() const
{
   cout << "\nDiameter = " << getDiameter()
        << "\nVolume = " << getVolume() << endl;
}  // end displayStatistics
```

Classes as Function Parameter

- Class objects can be passed to another function as parameters
- 3 methods of passing class as parameter to function
  - Pass by value
  - Pass by reference
  - Pass by const reference
- Pass by value – Any change that the function makes to the object is not reflected in the corresponding actual argument in the calling function.

Pass by value: Example

```
class subject
{
private:
        char subjectName[20];
        char kod[8];
        int credit;
public:
        subject (char *,char *,int k=3);
        void getDetail();
        friend void changeSubject(subject);
};
subject:: subject (char *sub,char *kd,int kre)
{       strcpy(subjectName,sub);
        strcpy(kod,kd);
        credit = kre;
}
void subject:: getDetail()
{
cout << "\n\nSubject Name  : " << subjectName;
cout << "\nSubject Code   : " << kod;
cout << "\nCredit hours   : " << credit;
}
```

Have to use friend function to pass object as parameter. This nonmember function is accessing private data member.

Pass by value: Example

```
// friend function implementation that receive object as parameter
void changeSubject(subject sub); // receive object sub
{ cout << "\nInsert new subject name: ";
  cin >> sub. subjectName;
  cout << "\nInsert new subject code: ";
  cin >> sub.kod;
  cout << "\n Get new information for the subject.";
  sub. getDetail();
}
main()
{ subject DS("Data Structure C++","SCJ2013");
  DS.getDetail();
  changeSubject(DS); // pass object DS by value
  cout << "\n View the subject information again: ";
  DS.getDetail();  // the initial value does not change
  getch();
};
```

Access class member, including private data member from sub.

Pass by reference

- Any changes that the function makes to the object will change the corresponding actual argument in the calling function.
- Function prototype for function that receive a reference object as parameter: use operator &

```
functionType functionName(className & classObject)
{
    // body of the function
{
```

Pass by reference - Example

```
// pass by reference
// friend function that receive object as parameter
void changeSubject(subject &sub);  // operator & is used
{ cout << "\nInsert new subject name: ";
  cin >> sub. subjectName;
  cout << "\nInsert new subject code: ";
  cin >> sub.kod;
  cout << "\n Get new information for the subject.";
  sub. getDetail();
}
main()
{ subject DS("Data Structure C++","SCJ2013");
  DS.getDetail();
  changeSubject(DS); // pass by reference
  cout << "\n View the subject information again: ";
  DS.getDetail();  // the value within the object has changed
  getch();
};
```

Const parameter

- Reference parameter can be declared as const if we don't want any changes being done to the data in the function.
- Function prototype for function that receive a reference object as parameter.

```
functionType functionName(const className & classObject)
{
    // body of the function
{
```

Const parameter - Example

```
void changeSubject(const subject &sub);
// operator const and & is used
{ cout << "\nInsert new subject name: ";
  cin >> sub. subjectName;
  cout << "\nInsert new subject code: ";
  cin >> sub.kod;
  cout << "\n Get new information for the subject.";
  sub. getDetail();
}
```

In this case, data member for sub is trying to be changed.
Error will occur since parameter const cannot be modified.

Const as return value from function

- Syntax for declaring function that return a class object

```
className functionName(parameter list)
{
    // function body
}
```

- Syntax to call function that return a class

objectName = functionName();

where,

- **objectName,** an object from the same class with the type of class return from the function. This object will be assigned with the value returned from function
- **functionName():** function that return class

Const as return value from function - Example

Function that return a class object, Point

```
Point findMiddlePoint(Point T1, Point T2)
{
    double midX, midY;
    midX = (T1.get_x() + T2.get_x()) / 2;
    midY = (T1.get_y() + T2.get_y()) / 2;
    Point middlePoint(t midX, midY);
    return middlePoint;
}
```

Function type is a class

Create instance of Point

Return instance of Point

Statement that call function that return a class

```
Point point1(10,5), point2(-5,5);
Point point3; // use defult argumen
point3 = findMiddlePoint(point1, point2)
// point3 is the point in the middle of point1 and point2
```

Call function that return object, value returned is assigned to point3

Array of class

- A group of objects from the same class can be declared as array of a class
- Example:
  - Array of class students registered in Data Structure Class
  - Array of class lecturer teaching at FSKSM
  - Array of class subjects offered in Semester I
- Every element in the array of class has it's own data member and function member.
- Syntax to declare array of objects :

```
className arrayName[arraySize];
```

Array of class - Example

```
class staff {
    char name[20];
    int age ;
    float salary;
public:
    void read_data() ;
    { cin >> name >> age >> salary;
    void print_data()
    { cout << name << age << salary; }
} ;

main()
{
        staff manager[20];
        // declare array of staff
}
```

Declare 20 managers from class staff. Each element has name, age and salary.

Array of class - Example

2 methods to call member function for `manager` array.

1. By using array subscript in order to access manager in certain location of the array.

```
cin >> n ;
manager[n].read_data() ;
cout << manager[n].name << manager[n].age ;
manager[n].print_data() ;
```

- 2. By using loop in order to access a group of managers

```
// read information for 10 managers
for ( int x = 0 ; x < 10; x++ )
    manager[x].read_data() ;
// print information of 10 managers
for ( int y = 0 ; y < 10; y++ )
    manager[y].print_data()
```

Pass array of object to function

Example

```
class info {
private:
   char medicine[15];
   char disease[15];
public:
void setMed() { cin >> medicine;}
void setDisease() { cin >> disease;}
char*getMedicine(){return medicine;}
char* getDisease() {return disease;}
};
```

Class info store information about disease and the relevant medicine

```
main()
{
info data[10];
for (int n = 0; n < 5; n++)
{
   data[n].setMedicine);
   data[n].setDisease();
}
cout <<"\nList of disease and
medicine";
for (int n = 0; n < 5; n++)
 cout << "\n" <<
data[n].getMedicine() <<
data[n].getDisease();
// pass the whole array to function
 checkMedicine(data);
getch();
}
```

Pass array of object to function

Example

- `checkMedicine(data)` function receive an array of object info. This function require the user to enter the name of the disease. Based on the disease's name, the function will search for the medicine that is suitable for the disease.

Pass array of object to function

```
void checkMedicine(info x[])
{ char diseas[20];
  int found = 0;
  cout << "\nEnter the disease name: ";
  cin >> diseas;
for (int n = 0; n < 5; n ++)
if (strcmp(diseas, x[n].getDisease()) == 0 )
{ cout << "\nMedicine for your disease: " << diseas << " is "
<< x[n].getMedicine();
found = 1;
break;
}
if (found == 0)
cout << "\nSorry, we cannot find the medicine for your disease.
Please refer to other physician.";
}
```

`checkMedicine(`**data);**
Call this function, where data is an array of objects from class info.

Pointer to object

- Pointer – store address of a variable.
- Pointer can also store address of an object.
- Example

```
student student1; // create instance of student
student* studentPtr = &student1;
```

- Create a pointer variable **`studentPtr`** and initialize the pointer with the address of instance **`student1`**

Pointer to object

Method to access class member through pointer variable **`studentPtr`** :

1) (***`studentPtr`**).print()

or

2) **`studentPtr`** ->print()

Pointer to object - Example

```
// pointer to object
#include <iostream.h>
#include <string.h>
class student
{
private:
char name[30];
unsigned long metricNo;
public: // konstruktor 2 param
student(char* nama,unsigned long num)
{
   no_metrik = num;
   strcpy(name, nama);
}
void print()
{ cout <<"\nStudent's name:" << name;
  cout <<"\nStudent's metric number:"
       << metricNo;
}
};
```

```
void main()
{
  student student1("Ahmad", 123123);
  student student2("Abdullah", 234234);
  cout << "Address of the object";
  cout << "\nAddress student1: "
       << &student1
       << "\nAddress student2 : "
       << &student2;
  student* ptr;
  cout << "\n\nPointer value ";
  ptr = &student1;
  cout <<"\nPointer value for student1"
       << ptr;
  ptr = &student2;
  cout <<"\nPointer value for student2"
       << ptr;
  ptr ->print();
}
```

Pointer to object

PROGRAM OUTPUT

```
Address of the object
Address student1: :0x0012ff68
Address student2: :0x0012ff44
Pointer value
Pointer value for student1:0x0012ff68
Pointer value for student2:0x0012ff44
Student's name: Abdullah
Student's metric number: 234234
```

Pointer to object

- We can also allocate memory for a pointer variable using operator new
- We can destroy memory for a pointer variable using operator delete
- Example:

```
void main()
{
    student *ptr = new student("Ahmad", 123123);
    ptr -> print();
    delete(ptr);
    ptr = new student("Abdullah", 234234);
    ptr ->print();
    delete(ptr);
}
```

Header File and Implementation File

- To implement ADT C++ separate files into header files and implementation files.
- This way, programmers can use implementation file without knowing how the member functions are implemented.
- Each class definition is placed in a header file
  - *Classname*.h
- The implementation of a class's methods are placed in an implementation file
  - *Classname*.cpp

The header file – Sphere.h

```
/** @file Sphere.h */
const double PI = 3.14159;
class Sphere
{
public:
  Sphere();                          // Default constructor
  Sphere(double initialRadius); // Constructor
  void setRadius(double newRadius);
  double getRadius() const;  // can't change data members
  double getDiameter() const;
  double getCircumference() const;
  double getArea() const;
  double getVolume() const;
  void displayStatistics() const;
private:
 double theRadius;          // data members should be private
};  // end Sphere
```

The implementation file – Sphere.cpp

```
/** @file Sphere.cpp */
#include <iostream>
#include "Sphere.h"   // must include header file
using namespace std;
Sphere::Sphere() : theRadius(1.0)
{
}  // end default constructor

Sphere::Sphere(double initialRadius)
{
   if (initialRadius > 0)
      theRadius = initialRadius;
   else
      theRadius = 1.0;
}  // end constructor
```

The implementation file – Sphere.cpp

```
void Sphere::setRadius(double
  newRadius)
{
    if (newRadius > 0)
        theRadius = newRadius;
    else
        theRadius = 1.0;
}   // end setRadius
```

- The constructor could call `setRadius`

The implementation file – Sphere.cpp

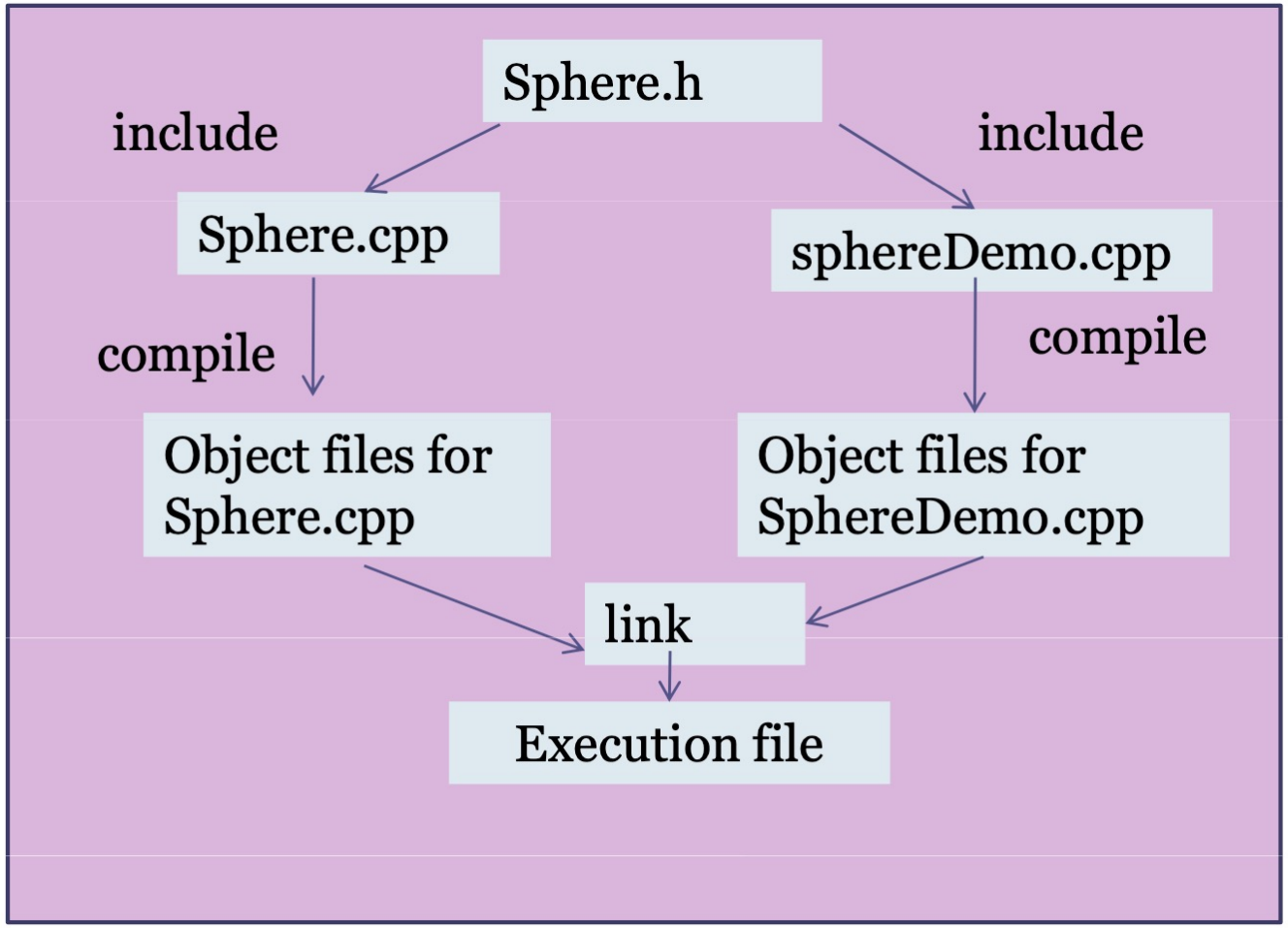
```
double Sphere::getRadius() const
{
   return theRadius;
}  // end getRadius
. . .

double Sphere::getArea() const
{
   return 4.0 * PI * theRadius *
  theRadius;
}  // end getArea
. . .
```

Client File: Using the class Sphere

```
/* SphereDemo.cpp */
#include <iostream>
#include "Sphere.h"   // must include header file
using namespace std;
int main()  // the client
{
   Sphere unitSphere;
   Sphere mySphere(5.1);
   cout << mySphere.getDiameter()
        << endl;
   . . .
}  // end main
```

File Compilation and Execution



Compile all .cpp files separately in order to create object files. Link all files to create .exe files.

Create File Project in Borland C++ Environment

Steps to create project file:

- Compile .cpp files separately.
- Crete project file by choose the menu
  <File> <New><Project>
- Set the project setting as follows:
  Project name : ProjectName.prj *or* **ProjectName.ide**
  Target name: *will create automatically*
  Target type: Application [.exe]
  Platform : Win32
  Target model : Console
  Frameworks/Control: Uncheck all the check boxes.
  Libraries: Static
  Then click **OK.**

Create File Project in Borland C++ Environment

Choose the implementation file for the projects:

- Right click the mouse button on the ProjectName.exe .
- Delete all files in the list.
- Choose ***<add node>*** and choose **Sphere.cpp** and
- Click ***<Open>*** button.
- Choose ***<add node>*** and choose `SphereDemo.cpp` and
- Click ***<Open>*** button.

- Link all object files by clicking ProjectName.exe and choose menu ***<Project> <Compile>***
  then choose
  <Project><Make all>.
- If there is a linker error, debug the error and link the files again.
- The execution file will be generated when all errors have been debug.
- Run the execution file by choosing ***<Debug>*** and ***<Run>.***

Conclusion

- In this class you have learned about: • Data Abstraction Concept
  - Review C++ concept:
    - Declaring a class, data member and function member
    - Creating constructor and destructor
    - Create object as function parameter
    - Return object from a function
    - Array of class & Pointer to class
    - Define and implement a class within header files and implementation files
- Welcome to Data Structure class

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

Thank You

innovative • entrepreneurial • global