SCJ2013 Data Structure & Algorithms

Introduction to Abstract Data Type & C++

Nor Bahiah Hj Ahmad & Dayang Norhayati A. Jawawi

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

Abstraction

Abstract data type (ADT)

- A collection of data and a set of operations on the data
- Given the operations' specifications, the ADT's operations can be used without knowing their implementations or how data is stored,

Abstraction

- The purpose of a module is separated from its implementation
- Specifications for each module are written before implementation

Abstraction

Data abstraction

- Focuses on the operations of data (*what* you can do to a collection of data), not on the implementation of the operations (*how* you do it)
- develop each data structure independently from the rest of the solution

Functional abstraction

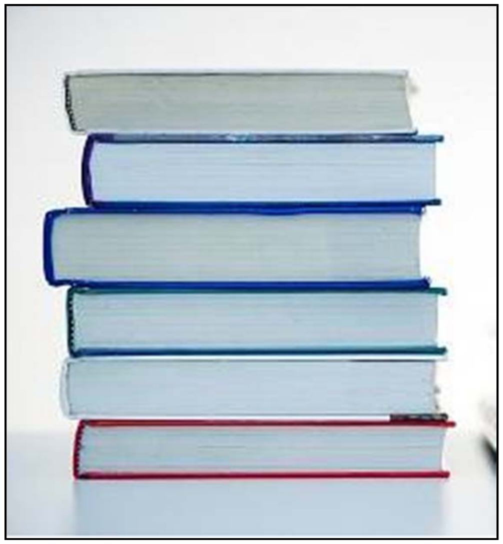
- Separates the purpose of a module from its implementation

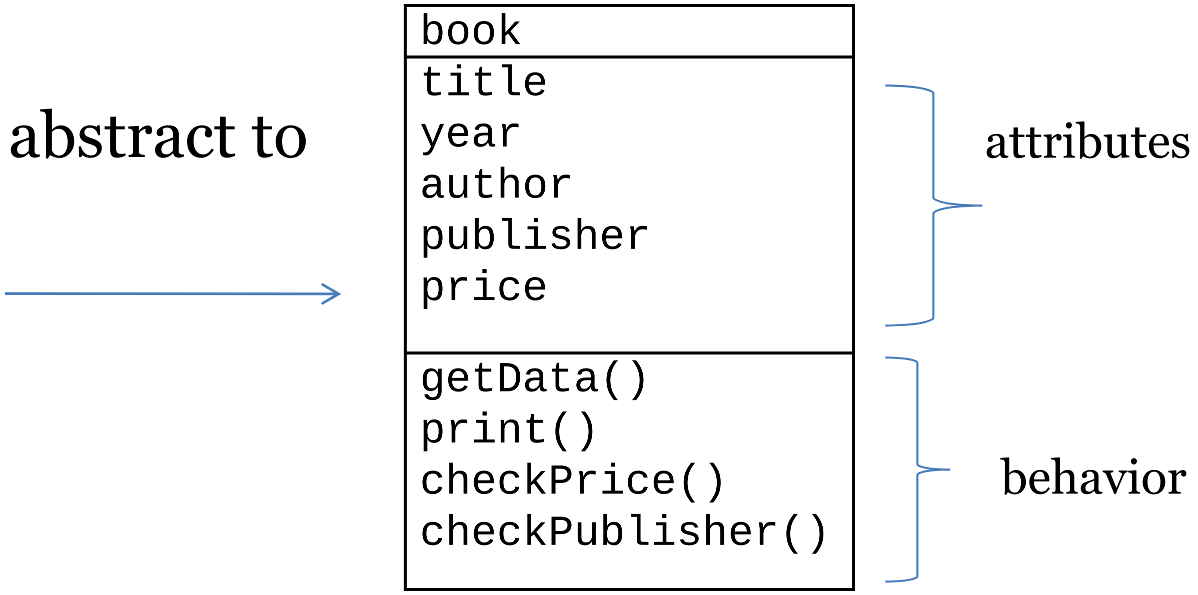
Information Hiding

Information hiding

- Hide the details within a module.
- To limit the way to deal with module and data, so that other module cannot modify the data.
- Makes these details inaccessible from outside the module.

Abstraction Example





Abstraction of a book

Encapsulation

- 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 is an important concept of data hiding.

C++ Classes

- 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
- An object is an instance of a class

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
```

$\Bigg\}$ class member declarations: data member and function member

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.

Class Definition for Book

```
class book
{ private:
  // data member declaration as private
    float price;
    int year;
    char author[20], title[25];
  public:
    book();      // Default constructor
    // Constructor with parmeter
    book(char *bkTitle,double bkPrice);
    book(int = 2000);
    // C++ function
    void getData();
    void print( );
    float checkPrice( )const;
    char * getAuthor();
    ~book() ;   // destructor
};  // end book declaration
```

Class Methods

Class methods consists of

- Constructor
- Destructor
- C++ functions.
- `const` function

Constructors

- Constructors
  - Used to create and initialize new instances of a class
  - Is invoked when an instance of a class is declared
  - Have the same name as the class
  - Have no return type, not even `void`
- A class can have several constructors
  - However, compiler will generate a default constructor if no constructor is defined.

Constructor Properties

- More than one constructor can be declared
- Each constructor must be distinguished by the arguments.

```
book();
book(char *bkTitle,double bkPrice);
book(int = 2000);
```

- Default constructor: `book();`
- Can have argument:

```
book(char *bkTitle,double bkPrice);
```

- Can have default argument:

```
book(int = 2000);
```

Default Constructor Implementation

- Sets data members to initial values

```
book::book()
{   price = 10.00;
    strcpy (author,”Dayang Norhayati”);
    strcpy (title, “Learn Data Structure”);
    year = 2012;
}   // end default constructor
```

Instance declaration:

```
book myBook;
```

Instance myBook is created with the price set to 10.0, author set to Dayang Norhayati, title set to Learn Data Structure and year set to 2012

Constructor with Argument Implementation

```
book::book (char *bkTitle,double bkPrice)
{    strcpy (title, bkTitle);
     price = bkPrice;
}
```

Instance declaration:

```
        book myBook(“NorBahiah”,25.00);
```

Price is set to **25.00**
Author is set to **NorBahiah**

Constructor With Default Argument Implementation

```
book::book(int year);
// Constructor with default argument
{  price = 10.00;
   strcpy (author,"NorBahiah");
   strcpy (title, "Learn C++");
}  // end default constructor
```

2 methods of to declare instance of a class:

```
book myBook; // set year to default value, 2000
book yourBook(2009); // set year to 2009
```

Avoid ambiguity error - when implementing constructor with default argument

Destructor

- Destroys an instance of an object when the object’s lifetime ends
- Each class has one destructor
  - The compiler will generate a destructor if the destructor is not defined
- Example: `~book();`

```
book::~book()
{ cout << "\nDestroy the book with title "
        << title;
}
```

Function Member Implementation

```
void book::getData()
{ cout << "\nEnter author's name : ";
  cin >> author;
  cout << "\nEnter book title : ";
  cin >> title;
}
```

Method to call the member function:

- From `main()` or non-member function

```
  cout << myBook.getData() << endl;
```

const member function – cannot alter value

```
    float book::checkPrice( )const
    {   return price;  }
```

Classes as Function Parameters

- 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

```
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;
}
```

friend function is used to pass object as parameter and allow non-member function to access private member.

Pass by value Continued…

```
// 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

```
// 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

```
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 the example, data member for sub is trying to be changed.
- Error will occur since parameter **`const`** cannot be modified.

Class 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

Class as Return Value from Function

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(midX, midY);
    return middlePoint;
}
```

Return 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 deful t argumen
// point3 is the point in the middle of point1 and point2
point3 = findMiddlePoint(point1,point2)
```

Call `findMiddlePoint` that return object and assign 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

```
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 of manager has name, age and salary.

Array of class

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

```
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;}
};
```

Declaration of class info that store information about the disease and the relevant medicine

Pass Array of Object to Function

```
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);
}
```

Function **checkMedicine(data)** receives an array of object **info**. This function requires the user to enter the name of the disease and the function will search for the medicine that is suitable for the disease.

Pass Array of Object to Function

From `main()`, statement `checkMedicine(data);`
calls this function, where `data` is an array of objects from class info.

```
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.";
}
```

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

2 methods to access class member through pointer variable **`studentPtr`** :

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

   or

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

Pointer to Object

```
// pointer to object
#include <iostream.h>
#include <string.h>
class student
{
private:
  char name[30];
  unsigned long metricNo;
public: // constructor
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;
}
}; // end of student class
```

```
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

- Operator **new** can also be used to allocate memory for a pointer variable.
- Operator **delete** destroys memory for a pointer variable.

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

Conclusion and Summary

- Abstract Data Type is a collection of data and a set of operations on the data.
- Abstraction implements information hiding and encapsulation, whereby other modules cannot tamper with the data.
- In C++, abstraction is implemented by using class.
  - In class declaration, there are declaration of data members and function members
  - Function members consists of constructor, destructor, c++ function and const function.
  - Object can be passed as function parameter by value or by reference.
  - Return type of a function can also be a class.
  - An Array and Pointer can also be declared of type class.

References

1. Nor Bahiah et al. *Struktur data & algoritma menggunakan C++. Penerbit UTM, 2005*
2. Richrd F. Gilberg and Behrouz A. Forouzan, "*Data Structures A Pseudocode Approach With C++*", Brooks/Cole Thomson Learning, 2001.