MODULE 2

ABSTRACT DATA TYPES AND C++

DATA STRUCTURE AND ALGORITHMS

FACULTY OF COMPUTING
UNIVERSITI TEKNOLOGI MALAYSIA

MODULE 2: ABSTRACT DATA TYPES AND C++

OBJECTIVES FOR STUDENTS

1. To introduce Abstract Data Type concept.
2. To review C++ programming on the following topics.

   - 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.
   - File operations.
   - Pointer to class.
   - Define and implement a class within header files and implementation files.

3. Able to write and debug programs in C++.

KEY CONCEPT

1.0 ABSTRCT DATA TYPE

1.1. **Abstraction**

- Separates the purpose of a module from its implementation.
- Specifications for each module are written before implementation.
- Two types of abstraction: functional abstraction and data abstraction.
- **Functional abstraction**
  - Separates the purpose of a module from its implementation

- **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.

1.2. 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.

1.3. Abstract data type (ADT)

- An ADT is composed of
  - A collection of data and 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
- We can use an ADT's operations without knowing their implementations or how data is stored, if you know the operations' specifications

1.4. Example of Abstraction

- Abstraction of a book, shown in Figure 2.1.
- **Data abstraction** - Think of features or attributes for books
  Example: book title, year published, the author, the publisher and the price.
- **Functional abstraction** – think of what we can do to the book.
  Example – get information of the book, print the information, check the price and check the publisher.

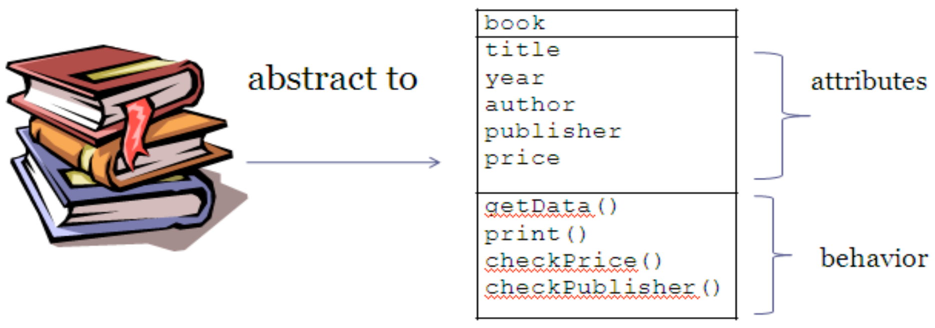


Figure 2.1 : Book Abstraction

1.5. **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 you can also specify them as public
- Encapsulation hides implementation details

2.0 C++ CLASS

2.1. Class definition

```
//Program 2.1
class clasName
{
public:
        list of data member declaration;
private:
        list of data member declaration;
};  // end class definition
```

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 member usually composed of data members and function members.

2.2. Example : Declaration of class book.

```
//Program 2.2
 class book
 {
   private:
   // data members should be private
     float price;
     int year;
     char author[20], title[25];
```

```
  public:
    book();      // Default constructor
    book(char *bkTitle,double bkPrice);
           // Constructor with parmeter
    book(int = 2020);
    // Constructor with default argumen
    void getData();
    void print( );
    float checkPrice( )const;
    char * getAuthor();
    ~book() ;   // destructor
};  // end book declaration
```

2.3. The book class consists of data members and function members. Function members operate on a collection of data, which is stored in a structured way in the computer memory.

2.4. 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
    - normal functions.

2.5. Constructor

- 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

2.6. Constructor Properties

- A class can have more than one constructor declaration, which is known as constructor overloading.
- Each constructor must be distinguished by the arguments as shown in Program 2.3.

```
//Program 2.3
book();
book(char *title,double price);
book(int = 2020);
```

Types of constructor declared in Program 2.3 are as follows:

- Default constructor, constructor without argument: **book();**
- Constructor with argument:
  book(char * bkTitle,double bkPrice);
- Constructors with default argument: **book(int = 2020);**

2.7. Default Constructor Implementation

- The implementation of a method qualifies its name with the scope resolution operator **::**
- Usually is used to initialize data members to certain values
- Program 2.4 is the implementation of default constructor for **book** class.

```
//Program 2.4
book::book()
{  price = 15.00;
   strcpy (author,"Dayang Norhayati");
   strcpy (title, "Learn Data Structure");
   year = 2014;
}  // end default constructor
```

- Instance declaration for the default constructor **book**:
  book myBook;
- Once the statement **book myBook;** is executed, the constructor in Program 2.4 will be invoked and the attributes for **myBook** will be assigned as follows:

Price : 15.00
Author : Dayang Norhayati
Title : Learn Data Structure
Year : 2014

2.8. Constructor with Argument Implementation

```
//Program 2.5
// Constructor with argument implementation
book::book (char *bkTitle,double bkPrice)
{   strcpy (title, bkTitle);
    price = bkPrice;
}
```

- The implementation of **book** constructor with two arguments is shown in Program 2.5.
- Instance declaration based on the constructor in Program 2.5:
  book myBook("Brain Power",25.00);
- The attributes for **myBook** will be assigned as follows:
  Price is set to **25.00**
  Title : Brain Power
- No values have been assigned for **Title** and **Year**

2.9. Constructor with Default Argument Implementation

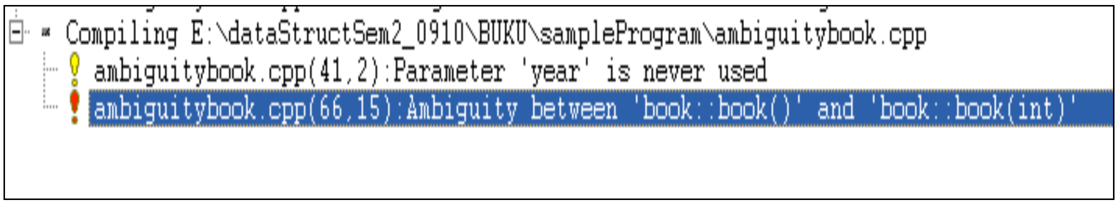
```
//Program 2.6
// Constructor with argument implementation // for
book(int = 2020);

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

- 2 methods to declare instance of a class:
  book myBook;// set year to default value, 2020
  book yourBook(2016); // set year to 2016

- However, when implementing overload constructor that consists of constructor with default argument, and default constructor as given in Program 2.7, you must avoid ambiguity error.

```
//Program 2.7
book();  //default constructor
book(int = 2020); // constructor with default
                  // argument
```

- The declaration : **book myBook;** gives ambiguity error as shown in Figure 2.2.
- The compiler cannot distinguish whether to invoke the default constructor or the constructor with default argument.



```
Compiling E:\dataStructSem2_0910\BUKU\sampleProgram\ambiguitybook.cpp
  ambiguitybook.cpp(41,2):Parameter 'year' is never used
  ambiguitybook.cpp(66,15):Ambiguity between 'book::book()' and 'book::book(int)'
```

Figure 2.2 : Ambiguity error

- Solution to avoid ambiguity error :
  - Discard one of the constructor or
  - Change the constructor with default argument from:

    book(int = 2000);
    to
    book(float p, int = 2000);
  - Program 2.8 shows valid overload constructor with different number and types of argument

```
//Program 2.8
book();
book(char *title,double price);
book(float p, int = 2020);
```

- Various instance declaration for the overload constructors in Program 2.8 are as follows:

  book aBook;
  book myBook("Fun Learning", 55.00);
  book labBook(25.00, 2014);
  book noteBook(25.00);

2.10. Destructor

- Destructor destroys an instance of an object when the object's lifetime ends
- Each class can only has one destructor
  - However, destructor can be omitted.
  - The compiler will generate a destructor if it is not defined.
- Destructor example:
  - **~book();**
- Destructor implementation

```
//Program 2.9
book::~book()
{ cout << "\nDestroy the book with title" << title;
}
```

2.11. Member Function Implementation

- Function implementation outside the class declaration is as follows:

```
//Program 2.10
void book::getData()
{ cout << "\nEnter author's name : ";
  cin >> author;
  cout << "\nEnter book title : ";
  cin >> title;
}
```

- **Const member function** – the function cannot alter value as shown in Program 2.11.

```
//Program 2.11
float book::checkPrice( )const
{
   return price;
}
```

- **Invoking the member function**:
  - **Member function can be invoked from main() or nonmember function**
  - The function must be invoked using the instance of the class.
  - Program 2.12 shows how member function **getData()**and **checkPrice( )**are invoked by **myBook** instance.

```
//Program 2.12
book myBook;   //declare book instance
myBook.getData();
cout << "\nThe book price is: RM:" <<
myBook.checkPrice( );
```

- **Invoke from member function of the same class** – the function can be invoked directly without using instance.
- The example in Program 2.13 invoke **getAuthor()**and **checkPrice()**from member function **print( )**.

```
//Program 2.13
void book::print( )
{ cout << "\nAuthor's name : " << getAuthor();
  cout << "\nBook title : " << title;
  cout << "\nBook price : " << checkPrice();
  cout << "\nBook published : " << year;
}
```

3.0 CLASS 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.

- Example of object as parameter and send using pass by value is shown in Program 2.14a. To allow the non-member function to access the private values declare the function as **friend**.
- The declaration: **friend void changePrice(book);** is a friend function that receive object book as parameter.
- The function can access class member, including private data member from book instance.

```
//Program 2.14a
// Class as function parameter
// pass by value
class book
{   private:
  // data members should be private
    float price;
    int year;
    char author[20], title[25];
  public:
    book();      // Default constructor
    :
   friend void changePrice(book);
   // non-member function that receives
   // class as parameter
};  // end book declaration
```

- The implementation of function **changePrice(book)**ia as shown in Program 2.14b.
- This function give discount to any book published on 2010 or earlier.

```
//Program 2.14b
// function that receive object book
void changePrice(book newBook)
{   if (newBook.year <=2010)
    { cout << "\nThe book has discount.";
      newBook.price -= 50.00/100 * newBook.price;
      cout << "\nThe new price after 50% discount
                 is RM" << newBook.price;
      newBook.print();
                //the price has been changed
    }
    else
      cout << "\nThe book has no discount.";
}
```

- Program 2.14c shows the example to call **changePrice()**function in Program 2.14b from main()
- Object **myBook** that is sent to function **changePrice()**only change the value in the function. However, the initial value of **myBook** is still retain when the program return to **main()**.

```
//Program 2.14c
//call function changePrice()from main()
main ( )
{  book myBook;  //invoke default constructor
     :
   changePrice(myBook);
   // pass objek myBook by value
   cout << "\n View the book information.";
   myBook.print();
  // the initial value does not change
}
```

- **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 need to use operator **&** as shown below:

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

- **Example of** pass by reference by using operator **&**:
  friend void changePrice(book &);
- **Need to use** friend function that receive object as parameter, in order to allow the function to access private member of class book.
- Program 2.15 show an example of pass by reference :

```
//Program 2.15
// Class as function parameter
// pass by reference
class book
{   private:
  // data members should be private
    float price;
    int year;
    char author[20], title[25];
  public:
    book();     // Default constructor
    :
   friend void changePrice(book &);
};  // end book declaration

// receive object book and use operator &
void changePrice(book& newBook)
{  :
   :
}

// Call from main()
main ( )
{        book myBook;
      // pass object myBook by reference
      :
      changePrice(myBook);
      myBook.print();
      // the attribute price has been changed
}
```

- **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 is as follows:

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

- Example program is in Program 2.16.

```
//Program 2.16
// operator const and & is used
void changePrice(const book& newBook)
{  if (newBook.year <=2005)
   { cout << "\nThe book has discount.";
     newBook.price -= 50.00/100 * newBook.price;
     // error, when try to change price value
```

```
        cout << "\nThe new price after 50%
          discount is RM" << newBook.price;
        newBook.print();
       }
   }
```

- In this example, the variable **price** for object **newBook** is trying to be changed. Error will occur since parameter **const** cannot be modified.

4.0 CLASS AS RETURN VALUE FROM FUNCTION

4.1. Function normally return value such as **int**, **float**, **char** etc.

4.2. For example, the following function : **float checkPrice( );** returns **float** value from the function.

4.3. However, function can also return class.

- Syntax for declaring function that return a class object

```
//Program 2.17
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

- Example of a function that return a class object, is shown in Program 2.18

```
//Program 2.18
book buyBook()
{  cout << "\Enter buyBook().";
   book aBook(200.00,2010);
   cout << "\Leaving buyBook().";
   return aBook;
}
```

- Statement that calls function that returns a class is shown in Program 2.19.

```
//Program 2.19
main ( )
{  book myBook;
   :
   book newBook;
   newBook = buyBook();
   //assign the return value to newBook
   newBook.print();
   return 0;
}
```

5.0 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 FC
  - Array of class subjects offered in Semester I.
- Every element in the array of class has its own data member and function member.
- Syntax to declare array of objects :
  className arrayName[arraySize];
- Example of array of class book

```
//Program 2.20
main ( )
{  // declare array of book with size 5
      book myBook[5];
   // Each element of the array has title
   //  author, price, and year.

    // To access only certain element in the
    // array, will access based on array index
    // Accessing book at index 0
           myBook[0].getData();
           myBook[0].print();

    // using loop to access the whole elements
    // in the array
    for (int i =0; i<5; i++)
    { myBook[i].getData();
      myBook[i].print();
    }
}
```

- 2 methods to call member function for **book** array in Program 2.20
- By using array subscript
  - Subscript can be used to access element of the array at certain index in the array.
  - Example:

```
cout << " Please enter the index of the array. ";
cin >> n ;
myBook[n].getData();
myBook[n].print();
```

- By using loop in order to access a group of elemts in the array

```
// read and print information for 5 books
for (int i =0; i<5; i++)
{    myBook[i].getData();
     myBook[i].print();
}
```

Pass array of object to function:

- Array can be passed as parameter to function. The whole elements in the array can be passed to other function or simply pass only one element from the array. Program 2.21a is an example of passing the whole array to function.

```
//Program 2.21a
void checkInfo(book x[])
{ int found = 0;
  char *authorName;
  cout << "\nEnter the author's name: ";
  cin >> authorName;
  for (int n = 0; n < 5; n ++)
    if (strcmp(authorName,
              x[n].getAuthor()) == 0 )
    { cout << "\nBook Title written by " <<
      authorName << "  " << x[n].getTitle();
      found = 1;
```

```
        break;
      }
   if (found == 0)
      cout << "\nSorry, we cannot find the
              book written by the author.";
}
```

- The header function **checkInfo(book x[]);** in Program 2.21a receive an array of object book. This function requires the user to enter the name of the author. The function will search from the book array for the title of the book based on the author's name and display the book that the author wrote.
- Program 2.21b shows the example to call function **checkInfo()** from **main()**. The statement **checkInfo(myBook);** in line 8 call function **checkInfo()**, and pass address of objects from **book** array .

```
1   //Program 2.21b
2   main ( )
3   {  book myBook[5];
4      for (int i =0; i<5; i++)
5      { myBook[i].getData();
6        myBook[i].print();
7      }
8      checkInfo(myBook);
9      // pass address of myBook to checkInfo()
10  }
```

6.0 POINTER TO OBJECT

6.1. Pointer to Object

- Pointer is used to store address of a variable.
- For ADT, pointer can also store address of an object.
- Example of pointer declaration:

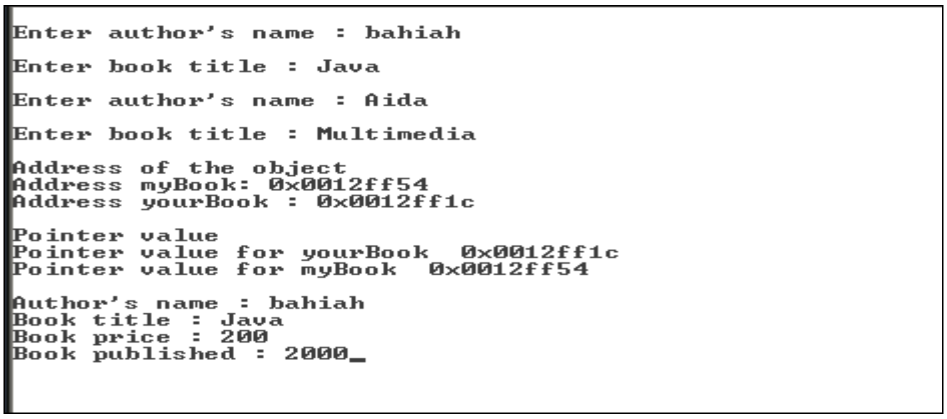
```
book textBook; // create instance of book
book *bookPtr ;  // pointer declaration
bookPtr = &textBook;  // assign address to pointer
```

- The codes in the example above declare pointer **bookPtr** that store address of **textBook** which is an instance b**ook** .
- Two methods to access class member through pointer variable **bookPtr** :
  1) **(*bookPtr).print()**
     or
  2) **bookPtr ->print()**

- Program 2.22 shows example of pointer to object implementation.

```
//Program 2.22
 // pointer to object
 main()
 { book myBook(200.00, 2000);
   book yourBook(50.00, 2010);
   cout << "\nAddress of the object";
   cout << "\nAddress myBook: " << &myBook
       << "\nAddress yourBook :"<< &yourBook;
   book* ptr;
   cout << "\n\nPointer value ";
   ptr = &yourBook;
   cout <<"\nPointer value for yourBook "<<ptr;
   ptr = &myBook;
   cout <<"\nPointer value for myBook "
       << ptr << "\n";
   ptr ->print();
 }
```

6.2. Output for Program 2.22

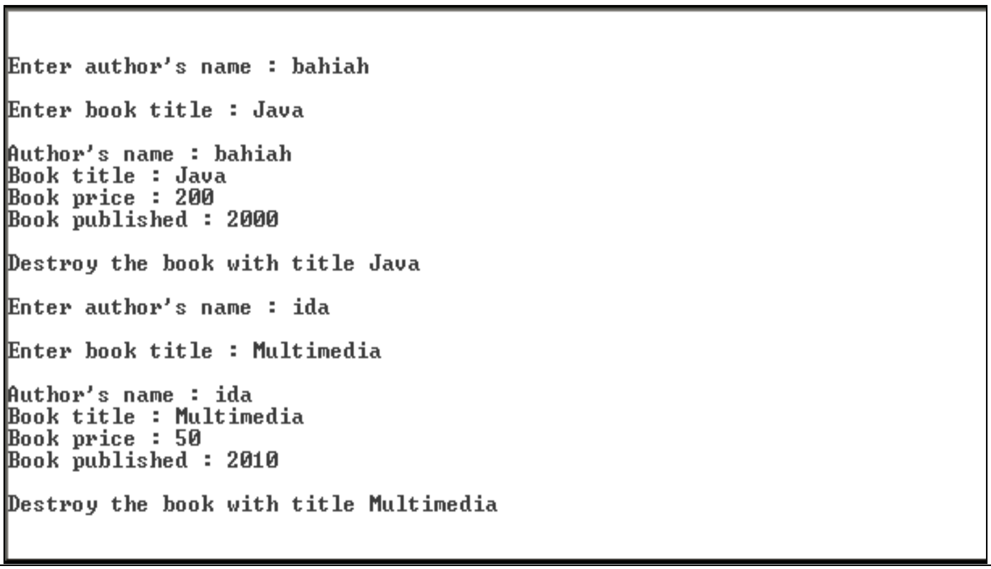


```
Enter author's name : bahiah
Enter book title : Java
Enter author's name : Aida
Enter book title : Multimedia

Address of the object
Address myBook: 0x0012ff54
Address yourBook : 0x0012ff1c

Pointer value
Pointer value for yourBook  0x0012ff1c
Pointer value for myBook  0x0012ff54

Author's name : bahiah
Book title : Java
Book price : 200
Book published : 2000_
```

- Memory for a pointer variable, can also be allocated dynamically using operator **new** and destroy the memory using operator **delete**
- Program 2.23 shows the example of using **new** and **delete** operator

```
//Program 2.23
 // pointer to object
 // new and delete pointer
main()
{
   book *ptr = new book(200, 2000);
   ptr -> print();
   delete(ptr);

   ptr = new book(50.00, 2010);
   ptr ->print();
   delete(ptr);
}
```

6.3. Output for Program 2.23



```
Enter author's name : bahiah

Enter book title : Java

Author's name : bahiah
Book title : Java
Book price : 200
Book published : 2000

Destroy the book with title Java

Enter author's name : ida

Enter book title : Multimedia

Author's name : ida
Book title : Multimedia
Book price : 50
Book published : 2010

Destroy the book with title Multimedia
```

7.0 BASIC C++ FILE OPERATIONS

7.1 Introduction

- Programs need data to be processed. The data can be key-in as input from the user or can be initialized in the program.
- However, usually programs need large data. The data can be stored in a **file** and saved in external storage. Data stored in a file can be accessed and modify by a program easily at any time.

7.2 Basic steps to process file in C++

1. Create the input file, usually as .txt file.
2. Include header file **fstream.h** in the program in order to use the file libraries.

```
# include <fstream.h>
```

3. Declare the object file as file input or output.

```
ifstream bookFiles;     // input file declaration
ofstream outFile;   // output file declaration
```

4. Open a file before any file operations can be processed. When a file is opened, we need to test whether the file has been opened successfully.

```
bookFiles.open("bookData.txt", ios::in);
if (!bookFiles)
{ cerr << "ERROR!!!!Cannot open the file.";
  return 0;
}
```

5. Implement file operations such as:
   Read from file: **bookFiles >> year >> price;**
   Getline() : **bookFiles.getline(title, 25, '\n');**
   Print to file: **outFile << year << price;**

6. Close file : **bookFiles.close();**

7.3 Creating Data File **bookData.txt**. The file consists of the year, price, title and author of 4 books.

```
2010   20.00 Learn Data Structure
Nor Bahiah Ahmad
2009   30.00 Advanced C++
Zalmiyah Zakaria
2005   45.00 Learn Java in 1 Day
Dayang Norhayati
2010   20.00 Data Structure and Algorithm
Aida Ali
```

7.4 Program 2.24 is an example program that open and access **bookData.txt** file.

```
//Program 2.24
// File operation
#include <string.h>
#include <iostream.h>
#include <fstream> // include header file

```

```
fstream bookFiles; // declare object file
class book
{
  private:
  // data members should be private
    float price;
    int year;
    char author[20], title[25];
  public:
    void getData();
    void print( );
}; // end book declaration

// function that read data from file
void book::getData()
{   bookFiles >> year >> price;
    bookFiles.getline(title, 25, '\n');
    bookFiles.getline(author, 20, '\n');
}

void book::print( )
{ cout << "\nAuthor's name : " << author;
  cout << "\nBook title : " << title;
  cout << "\nBook price : " << price;
  cout << "\nBook published : " << year;
}

main ( )
{   // open a file and test for error
    bookFiles.open("bookData.txt", ios::in);
    if (!bookFiles)
   { cerr << "ERROR!!!!Cannot open the file.";
     return 0;
   }

    book myBook[4];   //invoke default constructor

    for (int j=0;j<3;j++)
        myBook[j].getData();  // read from file
    for (int j=0;j<3;j++)
        myBook[j].print();

    bookFiles.close();  // close file
}
```

8.0 HEADER FILE AND IMPLEMENTATION FILE

8.1. To implement ADT C++, we need to separate C++ program 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**

8.2. For the **book** class example, we must separate the file into header file, named **book.h**, and the implementation file that contain the class member function source code, named **book.cpp** and another implementation file that contain source codes for **main()** program named, **bookMain.cpp**.

8.3. The header file – **book.h** is shown in Program 2.25a

```
//Program 2.25a
// Header file

/** @file book.h */
class book
{
  private:
  // data members should be private
    float price;
    int year;
    char author[20], title[25];
  public:
    book();   //default constructor
    book(char *bkTitle,double bkPrice);
          // Constructor with parmeter
    void print( );
    float checkPrice( )const;
    char * getAuthor();
};  // end book declaration


```

8.4. The implementation file for member function source codes - **book.cpp**

```
//Program 2.25b
// Implementation file
/** @file book.cpp */

#include <string.h>
```

```
#include <iostream.h>
#include <conio.h>
#include "book.h"   // must include header file

book::book()
 {  price = 10.00;
    strcpy (author,"Dayang Norhayati");
    strcpy (title, "Learn Data Structure");
    year = 2010;
}  // end default constructor
book::book (char *bkTitle, double bkPrice)
{    strcpy (title, bkTitle);
     price = bkPrice;
     year = 2009;
     strcpy (author,"NorBahiah");
}
void book::print( )
{ cout << "\nAuthor's name :" << getAuthor();
  cout << "\nBook title : " << title;
  cout << "\nBook price : " << checkPrice();
  cout << "\nBook published : " << year;
}
float book::checkPrice( )const
{ return price; }
char* book::getAuthor()
{ return author; }
```

8.5. Client File - file that use the class book, such as **main()** - **bookMain.cpp**.

```
//Program 2.25c
// Client file

/* bookMain.cpp */
#include <string.h>
#include <iostream.h>
#include <conio.h>
#include "book.h"  // must include header file

main ( )
{  book myBook;   //invoke default constructor
   myBook.print();
   getch();
   book storyBook("UPIN & IPIN Adventure",50.00);
   storyBook.print();
   cout << "\n\nThe book price is: RM: " <<
                        storyBook.checkPrice( );
   cout << "\nThe book author is:  " <<
                         storyBook.getAuthor( );
   return 0;
}
```

8.6. File Compilation and Execution

Compile all .cpp files separately in order to create object files. Link all files to create .exe files. Figure 2.3 shows the compilation and link of the .cpp files.

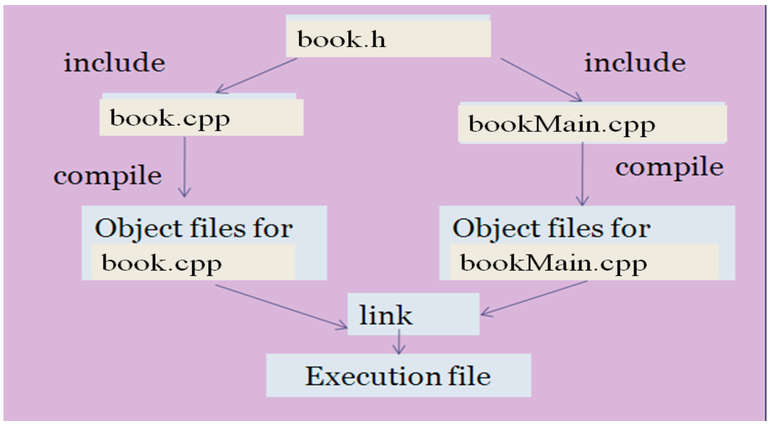


Figure 2.3 File compilation and executions

8.7. Create File Project in Borland C++ Environment

Steps to create project file:

1. Compile .cpp files separately.
2. Crete project file by choose the menu *<File> <New><Project>*
3. 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.**
4. 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 **book.cpp** and
   - Click ***<Open>*** button.
   - Choose ***<add node>*** and choose **book.Main** and
   - Click ***<Open>*** button.
5. 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.
6. Run the execution file by choosing <***Debug>*** and ***<Run>.***