CONFIDENTIAL

SCHOOL OF COMPUTING
Faculty of Engineering

UNIVERSITI TEKNOLOGI MALAYSIA

FINAL EXAMINATION (PRACTICAL)

SEMESTER II 2018/2019

SUBJECT CODE	**: SCSJ1023**
SUBJECT NAME	**: PROGRAMMING TECHNIQUE II**
YEAR/COURSE	**: 1 (SCSJ / SCSV / SCSB / SCSR / SCSP)**
TIME	**: 2.30 – 5.30 PM (3 HOURS)**
DATE/ DAY	**: 22nd MAY 2019**
VENUE	**: MPK 1 – 10, N28**

INSTRUCTIONS TO THE STUDENTS:

- This test consists of **TWO** questions. **ANSWER ALL QUESTIONS**.
- You are given **THREE HOURS** to complete the test **INCLUSIVE** the **submission of your program**.
- **IMPORTANT:**
  - Students who submitted a program of **Question 2** with **COMPILATION ERRORS** will obtain **ZERO MARK** for the Question 2.
  - All the **COMMENT STATEMENTS** in the submitted program **WILL NOT BE EVALUATED**.

MATERIAL FOR THE TEST:

- You are provided with two source code files, **`foodApp.cpp`** and **`xyzstore.cpp`**.
- Download the files (compressed in a RAR file named **`scsj1023-paper2.rar`**) from **http://penilaian.fc.utm.my/**.
- **IMPORTANT:** You **MUST extract** the RAR file into the local hard drive of your computer. **Do not edit the code directly** from the WinRAR.

SUBMISSION PROCEDURE:

- Only the source code files are required for the submission (i.e. the edited **`foodApp.cpp`** and **`xyzstore.cpp`**). You do not need to compress the files.
- Submit the source code files via the **http://penilaian.fc.utm.my/**.

(This question booklet consists of 11 pages INCLUDING this page)

Question 1 **[35 Marks]**

You are given Program 1 (**foodApp.cpp**) with 15 syntax and/ or logical errors. The program consists of three classes: **Food**, **Vegetable** and **CannedFood**. **Food** class is a superclass. **Vegetable** and **CannedFood** are subclasses to the **Food** class. The program implements inheritance and polymorphism concepts. Debug the errors, then compile and run the program. The program should produce the output as in **Figure 1**.

```
//Program 1
class Food
{
   string desc;
   double price;

   public:
      Food(string desc, double price) {
        this->desc = desc;
        this->price = price;
      }

      double getDesc() const { return desc; }
      double calcPriceInRinggit() const { return price * USTOMYR; }
      void displayInfo() {
        cout << fixed << setprecision(2)
             << "Price: USD" << price << endl
             << "Price converted to Malaysian = MYR"
             << calcPriceInRinggit << endl << endl;
      }
};

class Vegetable : class Food
{
   int weight;

   public:
      Vegetable(string desc, double price, int weight) {
        this->weight = weight;
      }

      double calcWeightInGram() const {
        return weight * POUNDTOGRAM;
      }

      virtual void displayInfo() {
        cout << "Food description: " << getDesc() << endl
             << "Weight in pound: " << weight << " pound" << endl
             << "Weight in gram: " << calcWeightInGram << " grams"
             << endl;
        displayInfo();
      }
};
```

```

class CannedFood : class Food
{
   string type, expDate;

   public:
      CannedFood(string desc, double price, string type, string
      expDate) {
        this->type = type;
        this->expDate = expDate;
      }

      virtual void displayInfo() {
        cout << "Food description: " << getDesc() << endl
             << "Canned Food Type: " << type << endl
             << "Expired date: " << expDate << endl;
        displayInfo();
      }
};

int main()
{
   Food *f[] = { new Vegetable("Broccoli", 1.6, 3),
                 new Vegetable("Tomato", 1.4, 5),
                 new CannedFood("Mushroom Soup", 5.78, "Soups",
                                "12/09/2020"),
                 new Vegetable("Cabbage", 0.7, 4.5),
                 new CannedFood("Sliced Yellow Cling Peaches", 9.58,
                                "Fruit", "01/02/2021")};

   for (int i = 0; i < sizeof(*f) / sizeof(*f[0]); i++)
   {
      cout << "Food #" << (i + 1) << endl;
      f[i].displayInfo();
   }

   return 0;
}
```

```
Food #1
Food description: Broccoli
Weight in pound: 3 pound
Weight in gram: 1360.77 grams
Price: USD1.60
Price converted to Malaysian = MYR6.61

Food #2
Food description: Tomato
Weight in pound: 5 pound
Weight in gram: 2267.95 grams
Price: USD1.40
Price converted to Malaysian = MYR5.78

Food #3
Food description: Mushroom Soup
Canned Food Type: Soups
Expired date: 12/09/2020
Price: USD5.78
Price converted to Malaysian = MYR23.87

Food #4
Food description: Cabbage
Weight in pound: 4 pound
Weight in gram: 1814.36 grams
Price: USD0.70
Price converted to Malaysian = MYR2.89

Food #5
Food description: Sliced Yellow Cling Peaches
Canned Food Type: Fruit
Expired date: 01/02/2021
Price: USD9.58
Price converted to Malaysian = MYR39.57
```

Figure 1: Output of program

Question 2 **[65 Marks]**

XYZ Pvt. Ltd. is a company that sells various item and has several stores throughout Malaysia. Every year, the HQ will collect sales data from all of its stores in paper form. The company is experimenting on the idea of using IT to record and store data in a file and hired a programmer to write a program. However, the programmer quits due to some unknown issue and you are tasked to complete his work.

The program was coded in C++ (see **xyzstore.cpp**) and contains two classes: **StoreManager** and **StoreData**. The relationship of the two classes is shown in **Figure 2**.

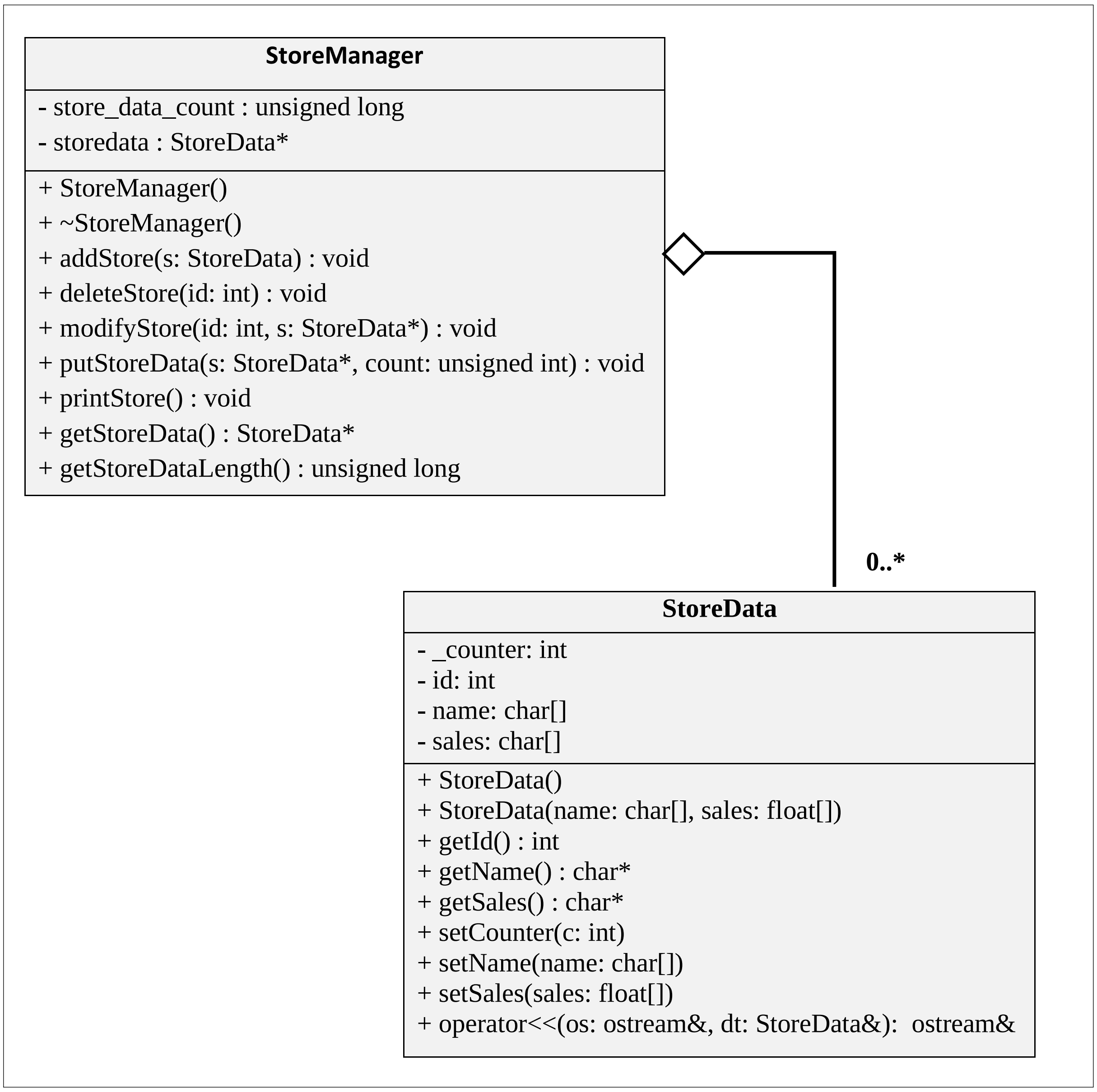
StoreManager
- store_data_count : unsigned long
- storedata : StoreData*
+ StoreManager()
+ ~StoreManager()
+ addStore(s: StoreData) : void
+ deleteStore(id: int) : void
+ modifyStore(id: int, s: StoreData*) : void
+ putStoreData(s: StoreData*, count: unsigned int) : void
+ printStore() : void
+ getStoreData() : StoreData*
+ getStoreDataLength() : unsigned long
0..*
StoreData
- _counter: int
- id: int
- name: char[]
- sales: char[]
+ StoreData()
+ StoreData(name: char[], sales: float[])
+ getId() : int
+ getName() : char*
+ getSales() : char*
+ setCounter(c: int)
+ setName(name: char[])
+ setSales(sales: float[])
+ operator<<(os: ostream&, dt: StoreData&): ostream&

Figure 2: UML class diagram

Below are details of each of the classes used in the program:

a) **StoreManager** class

   i) This class manages a dynamically allocated array of **StoreData** via '**storedata**'.

   ii) '**store_data_count**' keeps track of the number of **StoreData** pointed to by '**storedata**'.

   iii) '**addStore**' function adds **StoreData s** into the array '**storedata**'. The array is a dynamically allocated array. When a new store is added, a new memory allocation is made that could fit existing and the newly added store. Data from the previously allocated memory is copied to the newly allocated one. Once done, the previously allocated memory region is free-ed and the '**storedata**' is updated to point to the newly allocated memory.

   iv) '**deleteStore**' function deletes a **StoreData** entry pointed by '**storedata**' based on matching '**id**'. This is done by allocating a new memory region with one less space than the current one, copying all current **StoreData** to the newly allocated memory except the one with an **id** matching the one given in the parameter to the function **deleteStore**. Once done, the previously allocated memory region is free-ed and the '**storedata**' is updated to point to the newly allocated memory.

   v) '**modifyStore**' function updates '**name**' and '**sales**' of **StoreData** pointed by '**storedata**' based on the given parameter '**id**', with '**name**' and '**sales**' from **StoreData** in '**s**'.

   vi) '**putStoreData**' function updates '**storedata**' and '**store_data_count**' with '**s**' and '**count**'.

   vii) '**printStore**' function prints out data from each of the **StoreData** in the array pointed to by '**storedata**' (making use of the overloaded **operator<<** function).

   viii) '**getStoreData**' function is an accessor for '**storedata**'.

   ix) '**getStoreDataLength**' function is an accessor for '**store_data_count**'.

b) **StoreData** class

   i) This class store the '**name**' of a store and '**sales**' data (for 12 months) of a store.

   ii) There is a member called '**id**' which is set based on an internal counter '**_counter**'.

   iii) Function names starting with "**get**" are accessors while functions starting with "**set**" are mutators.

   iv) The class overloads **operator<<** to enable easy display of its internal data (via **cout**) such as '**id**', '**name**', and '**sales**'.

Based on the class diagram and description of classes given above, complete the program (**xyzstore.cpp**) that does the following tasks. ***IMPORTANT NOTE:*** **<u>Do not modify existing code</u>** in the template given.

Task 1: Write a default constructor for **StoreData** that sets or initializes **id**, **name**, and **sales** to 0 *(line 80).* (5 marks)

Task 2: Write an accessor function for **StoreData**'s **id** *(line 120).* (2 marks)

Task 3: Write an accessor function for **StoreData**'s **name** *(line 129).* (2 marks)

Task 4: Write an accessor function for **StoreData**'s **sales** *(line 139).* (2 marks)

Task 5: Write a mutator function for **StoreData**'s **_counter** *(line 149).* (2 marks)

Task 6: Write a mutator function for **StoreData**'s **name** *(line 159).* (3 marks)

Task 7: Write a mutator function for **StoreData**'s **sales** *(line 172).* (3 marks)

Task 8: Write an overloaded **operator<<** function to provide access to the value of **id**, **name**, and **sales** of **StoreData** *(line 184).* (5 marks)

Task 9: Write a destructor for **StoreManager** that will deallocate any allocated memory for **storedata**, if any *(line 216).* (3 marks)

Task 10: In '**addStore**' function *(line 228)* (9 marks)

- (a) : Allocate memory to **temp** to contain current and new **StoreData** *(line 235). (2 marks)*
- (b) : Copy existing **StoreData** to newly allocated space *(line 241). (2 marks)*
- (c) : Copy new **StoreData s** to end of newly allocated space *(line 250). (2 marks)*
- (d) : Update **store_data_count** to mark current size of **storedata** *(line 255). (1 mark)*
- (e) : Allocate memory for **StoreData** array of one element *(line 268). (2 marks)*

Task 11: Search for **StoreData** in **storedata** with **id** matching the one given in the <u>first parameter</u> (**id**), and set it's **name** and **sales** to the one in **second parameter**. Exit the function once done *(line 327).* (7 marks)

Task 12: In '**printStore**' function *(line 349)* (5 marks)

- (a) : Print "***No data to print!***" message if there is no **StoreData** added, and exit the function *(line 352). (3 marks)*
- (b) : Print all **StoreData** in **storedata** *(line 364). (2 marks)*

Task 13: Write an accessor function for **StoreManager**'s **storedata** *(line 377).* (2 marks)

Task 14: In standalone '**write**' function *(line 474)* (5 marks)

- (a) : Open **filename** for output in binary mode *(line 482). (1 mark)*
- (b) : Write all **StoreData** in **s** to the file *(line 487). (3 marks)*
- (c) : Close the opened file *(line 492). (1 mark)*

Task 15: In standalone '**load**' function *(line 498)* (10 marks)

- (a) : Open **filename** for input in binary mode *(line 509). (1 mark)*
- (b) : Display an error message and exit function if unable to open the file *(line 514).*

(2 marks)

(c) : Get size of the file and assign it to **file_length** *(line 523).* *(2 marks)*

(d) : Calculate the number of **StoreData** objects in the file *(line 529).* *(1 mark)*

(e) : Allocate memory to contain all **StoreData** objects *(line 534).* *(1 mark)*

(f) : Read all data from the file into the newly allocated memory *(line 539).* *(2 marks)*

(g) : Close the opened file *(line 545).* *(1 mark)*

Sample input/ output of the program is shown in the **Figure 3**. ***Note:*** The one in **bold** are keyboard input to the program. Make sure that your code matches the output/ input sample given.

```
Welcome to XYZ Pvt Ltd -=<[XYZ]>=-
[1] Add new sales data
[2] Modify sales data
[3] Delete sales data
[4] Displays sales data
-
[5] Write sales data to file
[6] Load sales data from file
-
[0] Exit
 Select task : 1
Enter store name: skudai
Enter sales data : 11 11 45 78 56 25 36 25 14 85 23 65


Welcome to XYZ Pvt Ltd -=<[XYZ]>=-
[1] Add new sales data
[2] Modify sales data
[3] Delete sales data
[4] Displays sales data
-
[5] Write sales data to file
[6] Load sales data from file
-
[0] Exit
 Select task : 1
Enter store name: lol
Enter sales data : 45 56 58 52 36 58 47 58 69 52 41 25


Welcome to XYZ Pvt Ltd -=<[XYZ]>=-
[1] Add new sales data
[2] Modify sales data
[3] Delete sales data
[4] Displays sales data
-
[5] Write sales data to file
[6] Load sales data from file
-
[0] Exit
 Select task : 4
```

```
Sales data (id, name, sales) :
[1]     skudai   11 11 45 78 56 25 36 25 14 85 23 65
[2]     lol      45 56 58 52 36 58 47 58 69 52 41 25


Welcome to XYZ Pvt Ltd -=<[XYZ]>=-
[1] Add new sales data
[2] Modify sales data
[3] Delete sales data
[4] Displays sales data
-
[5] Write sales data to file
[6] Load sales data from file
-
[0] Exit
 Select task : 2
Enter id of store to modify : 2
Enter new store name : mersing
Enter new sales data : 45 56 58 52 36 58 47 58 69 52 41 25


Welcome to XYZ Pvt Ltd -=<[XYZ]>=-
[1] Add new sales data
[2] Modify sales data
[3] Delete sales data
[4] Displays sales data
-
[5] Write sales data to file
[6] Load sales data from file
-
[0] Exit
 Select task : 4

Sales data (id, name, sales) :
[1]     skudai   11 11 45 78 56 25 36 25 14 85 23 65
[2]     mersing  45 56 58 52 36 58 47 58 69 52 41 25


Welcome to XYZ Pvt Ltd -=<[XYZ]>=-
[1] Add new sales data
[2] Modify sales data
[3] Delete sales data
[4] Displays sales data
-
[5] Write sales data to file
[6] Load sales data from file
-
[0] Exit
 Select task : 5
[SAVE] Enter filename : record.dat


Welcome to XYZ Pvt Ltd -=<[XYZ]>=-
[1] Add new sales data
[2] Modify sales data
[3] Delete sales data
[4] Displays sales data
-
[5] Write sales data to file
[6] Load sales data from file
```

```
-
[0] Exit
 Select task : 3
Enter id of store to Delete : 1


Welcome to XYZ Pvt Ltd -=<[XYZ]>=-
[1] Add new sales data
[2] Modify sales data
[3] Delete sales data
[4] Displays sales data
-
[5] Write sales data to file
[6] Load sales data from file
-
[0] Exit
 Select task : 4

Sales data (id, name, sales) :
[2]     mersing  45 56 58 52 36 58 47 58 69 52 41 25


Welcome to XYZ Pvt Ltd -=<[XYZ]>=-
[1] Add new sales data
[2] Modify sales data
[3] Delete sales data
[4] Displays sales data
-
[5] Write sales data to file
[6] Load sales data from file
-
[0] Exit
 Select task : 3
Enter id of store to Delete : 1
Error !!! Id 1 not found !


Welcome to XYZ Pvt Ltd -=<[XYZ]>=-
[1] Add new sales data
[2] Modify sales data
[3] Delete sales data
[4] Displays sales data
-
[5] Write sales data to file
[6] Load sales data from file
-
[0] Exit
 Select task : 3
Enter id of store to Delete : 2


Welcome to XYZ Pvt Ltd -=<[XYZ]>=-
[1] Add new sales data
[2] Modify sales data
[3] Delete sales data
[4] Displays sales data
-
[5] Write sales data to file
[6] Load sales data from file
-
```

```
[0] Exit
 Select task : 4
 No data to print !


Welcome to XYZ Pvt Ltd -=<[XYZ]>=-
[1] Add new sales data
[2] Modify sales data
[3] Delete sales data
[4] Displays sales data
-
[5] Write sales data to file
[6] Load sales data from file
-
[0] Exit
 Select task : 6
[LOAD] Enter filename : record.dat


Welcome to XYZ Pvt Ltd -=<[XYZ]>=-
[1] Add new sales data
[2] Modify sales data
[3] Delete sales data
[4] Displays sales data
-
[5] Write sales data to file
[6] Load sales data from file
-
[0] Exit
 Select task : 4

Sales data (id, name, sales) :
[1]     skudai   11 11 45 78 56 25 36 25 14 85 23 65
[2]     mersing  45 56 58 52 36 58 47 58 69 52 41 25


Welcome to XYZ Pvt Ltd -=<[XYZ]>=-
[1] Add new sales data
[2] Modify sales data
[3] Delete sales data
[4] Displays sales data
-
[5] Write sales data to file
[6] Load sales data from file
-
[0] Exit
 Select task : 0
Thank you ! :)
```

Figure 3: Sample input/ output of program