(COMP2011)[2016](s)final~=f0bidkdp^_68579.pdf downloaded by dgmwong from http://petergao.net/ustpastpaper/down.php?course=COMP2011&id=19 at 2020-10-28 05:40:07. Academic use within HKUST only.

COMP 2011 Final - Spring 2016 – HKUST

- Date: May 25, 2016
- Time Allowed: 3 hours, 8:30 am - 11:30 am
- Instructions:
    1. This is a closed-book examination.
    2. There are 9 questions on 27 pages (including the cover page).
    3. Write your answers in the space provided in black/blue ink. NO pencil please. Otherwise you are not allowed to appeal for any grading disagreements.
    4. All programming codes in your answers must be written in ANSI C++.
    5. For programming questions, you are NOT allowed to define additional helper functions or structures, nor global variables unless otherwise stated. You also can not use any library functions not mentioned in the questions.

Student Name	Solution
Student ID	comp2011
ITSC email	
Lecture and Lab Section	

I have not violated the Academic Honor Code in this examination (signature):_____________

Problem	Score / Max. score
1	/7
2	/12
3	/3
4	/7
5	/6
6	/10
7	/15
8	/15
9	/25
Total	**/100**

(COMP2011)[2016](s)final~=f0bidkdp^_68579.pdf downloaded by dgmwong from http://petergao.net/ustpastpaper/down.php?course=COMP2011&id=19 at 2020-10-28 05:40:07. Academic use within HKUST only.

Problem 1 True/False Questions *[7 marks]*

Indicate whether the following statements are true or false by circling T or F. You get 1.0 point for each correct answer, −0.5 for each wrong answer, and 0.0 if you do not answer.

T [✓ **F**] (a) A destructor can specify a return type as void, i.e.

```
void ~Person();
```

T [✓ **F**] (b) The following code is syntactically correct.

```
void fn() { cout << "test"; };
void fn(int a = 0) { cout << "test" << a; };
int main()
{
   fn();
   return 0;
}
```

T [✓ **F**] (c) The following code is syntactically correct.

```
double* realPtr;
long* integerPtr;
integerPtr = realPtr;
```

T [✓ **F**] (d) Executing the following code will never cause a program crash.

```
int* ptr1 = new int;
int* ptr2 = ptr1;
delete ptr1;
delete ptr2;
```

[✓ **T**] **F** (e) All variables in a C++ program must be declared before they can be used.

T [✓ **F**] (f)
```
int x = 0;
for (int x = 0; x < 7; x++);
cout << x << endl;
```

Executing the above code will be produce the output:

```
7
```

T [✓ **F**] (g) The following is syntactically correct:

```
int* x, y; x = y;
```

(COMP2011)[2016](s)final~=f0bidkdp^_68579.pdf downloaded by dgmwong from http://petergao.net/ustpastpaper/down.php?course=COMP2011&id=19 at 2020-10-28 05:40:07. Academic use within HKUST only.

Problem 2 Multiple Choice Questions *[12 marks]*

Circle the letter to the left of the correct answers. There is only one correct answer per question. Each question is worth 2 points.

I. Which of the following is not a characteristic of linked list?

(a) A linked list can continue to grow until memory is exhausted.

(b) A linked list is a dynamic data structure – the length of the list increased or decreases as necessary.

(c) A linked list is a linear collection of self-referential class objects. A self-referential class object contains members (links) that point to objects of the same class type.

(d) A linked list supports random access.

II. Given a structure named `Rug` which is defined as follows:

```
struct Rug
{
   int length;
   int width;
};
```

And a variable `MyCarpet` is declared as follows:

```
Rug MyCarpet;
```

Which of the following is legal?

(a) `Rug.width = 12;`

(b) `Rug.length = Rug.width;`

(c) `MyCarpet = 15;`

(d) `MyCarpet.width = 10;`

III. What is the output after executing the following code?

```
int x = 9, y = 11;
if (x < 10)
if (y > 10)
cout << "*****" << endl;
else
cout << "#####" << endl;
cout << "$$$$$" << endl;
```

(a)
```
*****
$$$$$
```

(b)
```
*****
```

(c)
```
#####
$$$$$
```

(d)
```
$$$$$
```

(COMP2011)[2016](s)final~=f0bidkdp^_68579.pdf downloaded by dgmwong from http://petergao.net/ustpastpaper/down.php?course=COMP2011&id=19 at 2020-10-28 05:40:07. Academic use within HKUST only.

IV. If an array `temp` is declared as:

```
double temp[4];
```

then `&temp[2]` represents:

(a) The second element in the array

(b) The third element in the array

(c) The address of the array

[✓ (d) The address of the third element in the array]

V. Given a function prototype:

```
void fn(char*, int = 0);
```

and the following variables declared:

```
int n = 123;
char a[10] = "comp2011";
char* b = a;
```

Which of the following is/are legal function call(s)?

(1) `fn();`

(2) `fn(a);`

(3) `fn(a, n);`

(4) `fn(b, n);`

(a) All of the above

[✓ (b) 2, 3 and 4]

(c) 4 only

(d) 3 and 4

VI. Given the following struct and class definitions:

```
struct Date
{
   int day;
   int month;
   int year;
};

class Item
{
   public:
      char name[10];
      Date* date;
      Item(const char* str, int n, int d, int m, int y);
   private:
      int id;
};
```

and the following statement:

```
Item item1("comp2011", 123, 25, 5, 2016);
```

(COMP2011)[2016](s)final~=f0bidkdp^_68579.pdf downloaded by dgmwong from http://petergao.net/ustpastpaper/down.php?course=COMP2011&id=19 at 2020-10-28 05:40:07. Academic use within HKUST only.

Which of the following statement(s) is/are legal?

(1) `cout << Item.name << endl;`

(2) `cout << item1.id << endl;`

(3) `cout << item1.name << endl;`

(4) `cout << item1.date->day << endl;`

(5) `cout << item1.date.day << endl;`

(a) 2 only

(b) 3 and 4 only

(c) 2, 3 and 5 only

(d) none of them

(COMP2011)[2016](s)final~=f0bidkdp^_68579.pdf downloaded by dgmwong from http://petergao.net/ustpastpaper/down.php?course=COMP2011&id=19 at 2020-10-28 05:40:07. Academic use within HKUST only.

Problem 3 Pointer *[3 marks]*

What is the output of the following program?

```
#include <iostream>
using namespace std;

void mysteriousFunction1(char* s1, const char* s2)
{
   while (*s1 != '\0')
      ++s1;
   for (; (*s2 != '\0') && (*s1 = *s2); s1++, s2++);
   *s1 = '\0';
}

void mysteriousFunction2(char* s1, const char* s2)
{
   for (s2++; *(s2+1) != '\0'; s1++, s2++)
      *s1 = *s2;
   *s1 = '\0';
}

int main()
{
   char str1[20] = "comp", str2[20] = "2011";
   mysteriousFunction1(str1, str2);
   mysteriousFunction2(str2, str1);
   cout << "str1=" << str1 << endl;
   cout << "str2=" << str2 << endl;

   return 0;
}
```

str1=comp2011
Str2=omp201

Answer: ____

Grading Scheme:
First line: 1 mark
Second line: 2 marks

(COMP2011)[2016](s)final~=f0bidkdp^_68579.pdf downloaded by dgmwong from http://petergao.net/ustpastpaper/down.php?course=COMP2011&id=19 at 2020-10-28 05:40:07. Academic use within HKUST only.

Problem 4 Recursion *[7 marks]*

(a) *[3 marks]* The binomial coefficient C(n, k) is referred to the number of ways of picking k unordered outcomes from n possibilities. It can be defined recursively as

C(n, 0) = 1,

C(n, n) = 1 and,

for 0 < k < n, C(n, k) = C(n-1, k-1) + C(n-1, k).

A function named `binomial()` takes two positive integer arguments, n and k (i.e. greater than or equal to zero) and returns the binomial coefficient C(n, k). You can assume the input arguments are always correct. The recursive C++ implementation of the algorithm is shown below:

```
int binomial(int n, int k)
{
   cout << "binomial(" << n << ", " << k << ")" << endl;
   if (k == 0 || k == n)
      return 1;
   else
      return binomial(n-1, k-1) + binomial(n-1, k);
}
```

What is the output when the following main function is executed?

```
int main()
{
   int result = binomial(4, 3);
   return 0;
}
```

```
binomial(4, 3)
binomial(3, 2)
binomial(2, 1)
binomial(1, 0)
binomial(1, 1)
binomial(2, 2)
binomial(3, 3)
```

Answer: ____

Grading Scheme: 0.4 marks each, full marks when all correct

(COMP2011)[2016](s)final~=f0bidkdp^_68579.pdf downloaded by dgmwong from http://petergao.net/ustpastpaper/down.php?course=COMP2011&id=19 at 2020-10-28 05:40:07. Academic use within HKUST only.

(b) *[4 marks]* Implement a recursive function `sumNum()` that takes a positive integer n as input argument and returns the sum of all the digits in the integer. For example, if n is 357, then the digits in the number are 3, 5, and 7. The sum of all the digits is equal to 3 + 5 + 7 = 15, i.e. the function call `sumNum(357)` will return 15.

```
int sumNum(int n)
{
   // Add your code here

   if (n < 10)   // 1 mark
      return n;   // 1 mark
   else
      return sumNum(n / 10) + n % 10;  // 2 marks

/* *
General guideline: * max -2 points for syntax errors. Repeated syntax errors
count only once.
*/

}
```

(COMP2011)[2016](s)final~=f0bidkdp^_68579.pdf downloaded by dgmwong from http://petergao.net/ustpastpaper/down.php?course=COMP2011&id=19 at 2020-10-28 05:40:07. Academic use within HKUST only.

Problem 5 Scope *[6 marks]*

```
#include <iostream>
using namespace std;

class Ghost
{
   public:
     Ghost()
     {
        strcpy(name, "");
        cout << "Ghost(" << name << ")" << endl;
     }
     Ghost(char n[])
     {
       strcpy(name, n);
       cout << "Ghost(" << name << ")" << endl;
     }
     ~Ghost()
     {
        cout << "~Ghost(" << name << ")" << endl;
     }
   private:
     char name[20];
};

class PacMan
{
   public:
      PacMan()
      {
         inky = new Ghost("Inky");
         pinky = NULL;
         cout << "PacMan()" << endl;
      }
      PacMan(Ghost* other)
      {
         inky = NULL;
         pinky = other;
         cout << "PacMan(other)" << endl;
      }
      ~PacMan()
      {
         if (inky != NULL)
            delete inky;
         cout << "~PacMan()" << endl;
      }

   private:
      Ghost blinky;
      Ghost *inky;
      Ghost *pinky;
};
```

(COMP2011)[2016](s)final~=f0bidkdp^_68579.pdf downloaded by dgmwong from http://petergao.net/ustpastpaper/down.php?course=COMP2011&id=19 at 2020-10-28 05:40:07. Academic use within HKUST only.

```
int main()
{
   PacMan pm1;
   Ghost* other = new Ghost("other");
   PacMan* pm2 = new PacMan(other);

   delete pm2;
   delete other;

   return 0;
}
```

What is the output of executing the above program?

```
Ghost()
Ghost(Inky)
PacMan()
Ghost(other)
Ghost()
PacMan(other)
~PacMan()
~Ghost()
~Ghost(other)
~Ghost(Inky)
~PacMan()
~Ghost()
```

Answer: ____

Grading Scheme: 0.5 marks each, full marks if all correct

(COMP2011)[2016](s)final~=f0bidkdp^_68579.pdf downloaded by dgmwong from http://petergao.net/ustpastpaper/down.php?course=COMP2011&id=19 at 2020-10-28 05:40:07. Academic use within HKUST only.

Problem 6 Simulating Card Shuffling using Queue *[10 marks]*

A common card shuffling technique is called the riffle shuffle, in which a deck of cards is divided into two halves. The top half of the deck is placed in the left hand, the remaining half of the deck is in the other hand, and cards are then alternatively interleaved from the right and left hands forming one deck of cards again.

For example, a deck originally arranged as {Diamond A, Diamond K, Club 10, Square J, Square 2, Heart Q, Club 6} would become {Diamond A, Square J, Diamond K, Square 2, Club 10, Heart Q, Club 6}.

Consider a deck of cards as a queue, you are asked to simulate this riffle shuffle technique using a queue ADT. It will spit the queue into two halves. The first half will be stored in one queue and the other half will be stored in another queue. Then, it will merge the two queues by alternating the cards in the two queues.

Below is the header file of an array implementation of queue of card object.

```
#include <iostream> /* File: card-queue.h */
#include <cstdlib>
using namespace std;

const int BUFFER SIZE = 100;

struct card
{
   char suit[10];
   char rank[3];
};

class card_queue // Circular queue
{
  public:
    // CONSTRUCTOR member functions
    card_queue(void); // Default constructor
    // ACCESSOR member functions:
    bool empty(void) const; // Check if the queue is empty
    bool full(void) const; // Check if the queue is full
    int size(void) const; // Give the number of data currently stored
    card front(void) const; // Retrieve the value of the front item
    // MUTATOR member functions
    void enqueue(card); // Add a new item to the back of the queue
    void dequeue(void); // Remove the front item from the queue
  private:
    card data[BUFFER_SIZE]; // Use an array to store data
    int num_items; // Number of items on the queue
    int first; // Index of the first item; start from 0

};
```

(COMP2011)[2016](s)final~=f0bidkdp^_68579.pdf downloaded by dgmwong from http://petergao.net/ustpastpaper/down.php?course=COMP2011&id=19 at 2020-10-28 05:40:07. Academic use within HKUST only.

Implement the function

```
void riffle_shuffle(card_queue& cards)
```

using the card_queue ADT above so that the card_queue object cards is riffle shuffled after the function is called. You are not allowed to define any additional object such as array except two card_queue objects and a few int variables. Below is an example of calling the function where the card_queue object q1 is riffle shuffled:

```
card_queue q1;
card cards[7] = {{"Diamond", "A"}, {"Diamond", "K"}, {"Club", "10"}, {"Square",
                 "J"}, {"Square", "2"}, {"Heart" ,"Q"},  {"Club", “6"}};
for (int i=0; i<7; i++)
   q1.enqueue(cards[i]);
riffle_shuffle(q1);
```

Complete the implementation here:

```
void riffle_shuffle(card_queue& cards)
{
   // Add your code here

   card_queue q1, q2;                         // creating 2 queues with cards: 1 mark
   int size=cards.size();
   for (int i=0; i<size/2; i++)           // adding cards to the first queue: 2 marks
   {
      q1.enqueue(cards.front());
      cards.dequeue();
   }
   for (int i=cards.size(); i>0; i--)   // adding cards to the second queue: 2 marks
   {
      q2.enqueue(cards.front());
      cards.dequeue();
   }
                                          // able to handle odd number of cards: 1 mark

   while (!q1.empty() || !q2.empty())
   {                              // alternating the cards from the two queues: 4 marks
     if (!q1.empty())
     {
        cards.enqueue(q1.front());
        q1.dequeue();
     }
     if (!q2.empty())
     {
        cards.enqueue(q2.front());
        q2.dequeue();
     }
   }

  /* * enqueue from q2 first, then q1 also correct */

   /* *
   General guideline: * max -2 points for syntax errors. Repeated syntax errors
   count only once.
   */
}
```

(COMP2011)[2016](s)final~=f0bidkdp^_68579.pdf downloaded by dgmwong from http://petergao.net/ustpastpaper/down.php?course=COMP2011&id=19 at 2020-10-28 05:40:07. Academic use within HKUST only.

Problem 7 Class *[15 marks]*

This question involves two classes of objects, class `Room` and class `Flat`, to represent the relationship between rooms and flats. There is at least one Room in any `Flat`. In `Flat`'s constructor, the number of Rooms must be specified, and the constructor will dynamically allocate the required number of `Room` objects. `Flat`'s destructor must also deallocate all the `Room` objects. Other requirements of the two classes are:

- A `Room` has two member variables, length and width, to store the dimensions of the `Room`, which is always rectangular shaped.
- A `Flat` has two member variables, `Rooms` and `numRoom`. `Rooms` is a dynamic array of `Room` objects where the size of it is stored in `numRoom`.
- All member variables in the two classes have to be private.
- A Room is added to a `Flat` by calling a member function addRoom().
- Appropriate constructors and destructors have to be provided.
- Appropriate accessor and mutator member functions have to be provided.

Below is a testing program "flat-main.cpp" of the two classes:

```
#include "flat.h"

int main()
{
   Room* room1 = new Room(80.5, 80.5);
   Room room2;
   room2.set(81, 110.5);
   cout << "Area of room1: " << room1->getArea() << endl;

   Flat flat(2);
   flat.addRoom(1, *room1);
   flat.addRoom(2, room2);
   delete room1;
   cout << "Total area of flat: " << flat.getTotalArea() << endl;

   return 0;
}
```

Here is its output.

```
Area of room1: 6480.25
Total area of flat: 15430.8
```

Define and implement the two classes `Room` and `Flat` such that the above main function can be executed successfully and display the above output. You may assume that the input arguments for the member functions are always given correctly.

(COMP2011)[2016](s)final~=f0bidkdp^_68579.pdf downloaded by dgmwong from http://petergao.net/ustpastpaper/down.php?course=COMP2011&id=19 at 2020-10-28 05:40:07. Academic use within HKUST only.

```
/* *
  General guideline: * max -2 points for syntax errors. Repeated syntax errors
  count only once.
*/
```

(a) *[6 marks]* Write a complete C++ definition of the two classes `Room` and `Flat`, which is assumed to be saved in a file called "flat.h".

```
#include <iostream>
using namespace std;
// Add your code here

        class Room      // 3 marks: 0.5 each member with public/private correct
        {                                                    // otherwise: 0 mark
          public:
            // 2 constructors:
            Room();
            Room(double, double);
            // or 1 constructor: Room(double = 0, double = 0);

            double getArea() const;
            void set(double, double);

          private:
            double length;
            double width;
        };

        class Flat      // 3 marks: 0.5 each member with public/private correct
        {                                                    // otherwise: 0 mark
          public:
            Flat(int);
            ~Flat();

            double getTotalArea() const;
            void addRoom(int roomIndex, const Room& room) const;
          private:
            Room* rooms;
            int numRooms;
        };
```

(COMP2011)[2016](s)final~=f0bidkdp^_68579.pdf downloaded by dgmwong from http://petergao.net/ustpastpaper/down.php?course=COMP2011&id=19 at 2020-10-28 05:40:07. Academic use within HKUST only.

(b) *[9 marks]* Write the implementation of all the member functions of the classes `Room` and `Flat`. They are assumed to be saved in a separate file called "flat.cpp".

```
#include "flat.h"
// Add your code here

 Room::Room()                                                   // 1 mark
 {
    length = width = 0;
 }


 Room::Room(double l, double w)                                 // 1 mark
 {
    length = l;
    width = w;
 }


 double Room::getArea() const                                   // 1 mark
 {
    return width*length;
 }


 void Room::set(double l, double w)                             // 1 mark
 {
    length = l;
    width = w;
 }



 Flat::Flat(int n)                                              // 1 mark
 {
    rooms = new Room[n];
    numRooms = 2;
                    // or:
                    // numRooms = 0;
                    // but then in add_room(), there should be a line:
                    // numRooms++;
 }


 Flat::~Flat()                                                  // 1 mark
 {
    delete [] rooms;
 }


 void Flat::addRoom(int roomIndex, const Room& room) const      // 1 mark
 {
    Rooms[roomIndex-1] = room;
 }
```

(COMP2011)[2016](s)final~=f0bidkdp^_68579.pdf downloaded by dgmwong from http://petergao.net/ustpastpaper/down.php?course=COMP2011&id=19 at 2020-10-28 05:40:07. Academic use within HKUST only.

```
    double Flat::getTotalArea() const                          //  2 marks
    {
      double total = 0.0;
      for (int i=0; i<numRooms; i++)
      {
         total += rooms[i].getArea();
      }
      return total;
}
```

(COMP2011)[2016](s)final~=f0bidkdp^_68579.pdf downloaded by dgmwong from http://petergao.net/ustpastpaper/down.php?course=COMP2011&id=19 at 2020-10-28 05:40:07. Academic use within HKUST only.

Problem 8 Class: IntegerList with Dynamic Array *[15 marks]*

In this question, we are implementing an ADT, `IntegerList`, to provide operations on a list of integer numbers. Each object of the class `IntegerList` can hold a unique list of integer numbers, e.g. {9, 25, 10, 13, 2}. The integer numbers are not stored in any particular order. The internal representation is a dynamic array of integers. Member functions are provided in creating, modifying and accessing the `IntegerList` objects.

Below is the class definition in the header file:

```
#include <iostream>
using namespace std;

class IntegerList
{
  public:
    // CONSTRUCTORS
    IntegerList() { numItems = capacity = 0; items = NULL; }
      // default constructor
    IntegerList(int);  // dynamically create the array
    IntegerList(const int[], int);  // dynamically create the array and
                                    // stores the integers given

    // DESTRUCTOR
    ~IntegerList();

    // ACCESSORS
    void print() const;  // print all the elements
    int size() const;  // return the number of elements stored
    bool isFull() const; // if the number of elements is the same as the
                    // capacity, return true; otherwise, returns false;
    bool exists(int) const;  // if the element exists in the list,
                             // return true; otherwise, return false
    IntegerList union(const IntegerList& other) const;
      // create and return a third IntegerList object containing all
      // distinct  elements from the current IntegerList object and the
      // other IntegerList object
    IntegerList intersection(const IntegerList& other) const;
      // create and return a third IntegerList object containing
      // elements belonging to both the current IntegerList object and
      // the other IntegerList object

    // MUTATORS
    void addItem(int); // check if the number exists or not,
      // if not, add it to the end of the array; otherwise, do nothing;
      // dynamically expand the items array if the current number of items
      // stored is the same as the capacity (memory allocated, i.e. number
      // of elements already allocated)
    void deleteItem(int n); // if n exists in the array, remove it from
       // the array

  private:
    int* items;    // a dynamic array for storing the integers
    int numItems;  // the number of integers stored
    int capacity;  // the maximum possible number of integers
                   // be stored (number of elements allocated)
};
```

(COMP2011)[2016](s)final~=f0bidkdp^_68579.pdf downloaded by dgmwong from http://petergao.net/ustpastpaper/down.php?course=COMP2011&id=19 at 2020-10-28 05:40:07. Academic use within HKUST only.

Below is the sample main function:

```
int main()
{
   IntegerList s1(2);
   s1.addItem(9);
   cout << "Add 9 to s1, s1 is full? " << s1.isFull() << endl;
   s1.addItem(5);
   cout << "Add 5 to s1, s1 is full? " << s1.isFull() << endl;
   s1.addItem(3);
   cout << "s1 = "; s1.print();
   cout << "; size = " << s1.size() << endl;

   int numbers[] = {1, 2, 3, 4};
   IntegerList s2(numbers, 4);
   cout << "s2 = "; s2.print();
   cout << "; size = " << s2.size() << endl;

   IntegerList s3 = s1.union(s2);
   cout << "s3 = "; s3.print();
   cout << "; size = " << s3.size() << endl;

   IntegerList s4 = s1.intersection(s2);
   cout << "s4 = "; s4.print();
   cout << "; size = " << s4.size() << endl;

   s2.deleteItem(3);
   cout << "After removing 3, s2 = "; s2.print();
   cout << "; size = " << s2.size() << endl;

   return 0;
}
```

Which gives the following output:

```
Add 9 to s1, s1 is full? 0
Add 5 to s1, s1 is full? 1
s1 = {9, 5, 3}; size = 3
s2 = {1, 2, 3, 4}; size = 4
s3 = {9, 5, 3, 1, 2, 4}; size = 6
s4 = {3}; size = 1
After removing 3, s2 = {1, 2, 4}; size = 3
```

You are asked to complete the implementation of some of the member functions below.

```
/* *
  General guideline: * max -2 points for syntax errors. Repeated syntax errors
  count only once.
*/
```

(a) *[3 marks]* The overloading constructor `IntegerList::IntegerList(const int numbers[], int s)` will appropriately and dynamically allocate the `items` array, then copy and store all s integers from the given array, `numbers`, to the `items` array. The members, `numItems` and `capacity`, will be initialized by the value in s.

(COMP2011)[2016](s)final~=f0bidkdp^_68579.pdf downloaded by dgmwong from http://petergao.net/ustpastpaper/down.php?course=COMP2011&id=19 at 2020-10-28 05:40:07. Academic use within HKUST only.

```
IntegerList::IntegerList(const int numbers[], int s)
{
   // Add your code here


  /* Answer 1: */
      for (int i=0; i<s; i++)                         // 1 mark
          addItems(numbers[i]);                       // 2 marks

  /* // Answer 2: can assume the items in numbers array are unique
     capacity = s;                                      // 0.5 marks
     numItems = s;                                      // 0.5 marks

     items = new int[numItems];                         // 1 mark
     for (int i=0; i<s; i++)                            // 1 mark
       items[i] = numbers[i];
   */



}
```

(b) *[2 marks]* The destructor `IntegerList::~IntegerList()` will deallocate the dynamic `items` array.

```
IntegerList::~IntegerList()
{
    // Add your code here

    if (items != NULL)
    {
       delete [] items;                 // 2 marks, no mark for: delete items;
       items = NULL;
    }
    capacity = numItems = 0;


}
```

(COMP2011)[2016](s)final~=f0bidkdp^_68579.pdf downloaded by dgmwong from http://petergao.net/ustpastpaper/down.php?course=COMP2011&id=19 at 2020-10-28 05:40:07. Academic use within HKUST only.

(c) *[4 marks]* The member function void `IntegerList::addItem(int n)` will first check if the integer number, `n`, exists in the current `IntegerList` object or not. If it does not exist, then add it to the end of the array; otherwise, do nothing. You may need to dynamically expand the `items` array (reallocating the memory) if the number of items stored is the same as the capacity (i.e. number of elements already allocated).

```
void IntegerList::addItem(int n)
{
   // Add your code here

    if (!exists(n))                                                // 0.5 marks
    {
       if (numItems == capacity)         // 2.5 marks for reallocating the memory
       {
          int* temp = new int[++capacity];
          for (int i=0; i<numItems; i++)
             temp[i] = items[i];
          if (!items)
             delete [] items;
          items = temp;
       }
       items[numItems++] = n;                                       // 1 mark
    }
```

(d) *[3 marks]* The member function void `IntegerList::deleteItem(int n)` will remove the integer number, `n`, from the `items` array.

```
void IntegerList::deleteItem(int n)
{
   // Add your code here
   if (exists(n))
   {
      int i=0;
      for (; items[i] != n; i++);                 // finding the number: 1 mark
      for (; i<numItems-1; i++)         // shifting the remaining items: 1 mark
         items[i] = items[i+1];
      numItems--;                                 // decrement numItems: 1 mark
   }
```

(COMP2011)[2016](s)final~=f0bidkdp^_68579.pdf downloaded by dgmwong from http://petergao.net/ustpastpaper/down.php?course=COMP2011&id=19 at 2020-10-28 05:40:07. Academic use within HKUST only.

```
    }
```

(e) *[3 marks]* The member function `IntegerList IntegerList::intersection(const IntegerList& other)` will create and return a third `IntegerList` object containing all common integer numbers from both the current `IntegerList` object and the `other IntegerList` object.

```
IntegerList IntegerList::intersection(const IntegerList& other) const
{
  // Add your code here

  IntegerList result(numItems);              // create a new list: 1 mark
                                             // or: IntegerList result;
  for (int i=0; i<numItems; i++)             // 0.5 marks
    if (other.exists(items[i]))              // 0.5 marks
      result.addItem(items[i]);              // 1 mark
  return result;

}
```

(COMP2011)[2016](s)final~=f0bidkdp^_68579.pdf downloaded by dgmwong from http://petergao.net/ustpastpaper/down.php?course=COMP2011&id=19 at 2020-10-28 05:40:07. Academic use within HKUST only.

Problem 9 Class: IntegerList with Linked List *[25 marks]*

In the previous question, we have implemented the `IntegerList` class using dynamic array. We now change the internal representation to linked list, e.g. 9 -> 25 -> 10 -> 13 -> 2.

A structure of a node in a linked list, called `ll_node`, is defined as follows to store an integer data:

```
struct ll_node
{
  int data;
  ll_node* next;
};
```

And the private member variables of the `IntegerList` class definition are modified. The class definition becomes:

```
class IntegerList
{
  public:
    // CONSTRUCTORS
    IntegerList();
    IntegerList(int[], int);

    // DESTRUCTOR
    ~IntegerList();

    // ACCESSORS
    void print() const;
    int size() const;
    bool isFull() const;
    bool exists(int) const;

    IntegerList union(const IntegerList& other) const;
    IntegerList intersection(const IntegerList& other) const;

    // MUTATORS
    void addItem(int);
    void deleteItem(int);

  private:
    ll_node* head;
    void moveToFront(int);
};
```

The main function remains unchanged. But we will assume the member function `isFull()` always returns `false`. The output became

```
Add 9 to s1, s1 is full? 0
Add 5 to s1, s1 is full? 0
s1 = {9, 5, 3}; size = 3
s2 = {1, 2, 3, 4}; size = 4
s3 = {9, 5, 3, 1, 2, 4}; size = 6
s4 = {3}; size = 1
After removing 3, s2 = {1, 2, 4}; size = 3
```

(COMP2011)[2016](s)final~=f0bidkdp^_68579.pdf downloaded by dgmwong from http://petergao.net/ustpastpaper/down.php?course=COMP2011&id=19 at 2020-10-28 05:40:07. Academic use within HKUST only.

```
/* * General guideline: * max -2 points for syntax errors. Repeated syntax errors
count only once. */
```

Part I. You now have to re-implement some of the member functions using linked list representation of the integer elements.

(a) *[2 marks]* The constructor `IntegerList::IntegerList(int numbers[], int s)` will create a linked list by inserting the number in the given array of integers. The number of integers in the given array, `numbers`, is specified in `s`.

```
 IntegerList::IntegerList(int numbers[], int s)
 {
    // Add your code here
     head = NULL;                                       // 1 mark
     for (int i=0; i<s; i++)                            // 2 marks
         addItem(numbers[j]);
     /* Alternative solution: can assume all items in numbers array are unique
     // Copy the first node
     if (s == 0) { head = NULL; return; }                         // 0.5 marks
     ll_node* temp = head = new ll_node;
     temp->data = (numbers[0]);
     // Add a new ll_node for each of the remaining nodes
     for (int i = 1; i<s; i++, temp = temp->next)                   // 1 mark
     {
           temp->next = new ll_node;
           temp->next->data = (numbers [i]);
           temp->next->next = NULL;
     }
     temp->next = NULL; // Set the last ll_node to point to NULL  // 0.5 marks
     */
 }
```

(b) *[3 marks]* The destructor `IntegerList::~IntegerList()` will deallocate the dynamic linked list.

```
IntegerList::~IntegerList()
{
   // Add your code here
    ll_node* next;
    for (ll_node* temp = head; temp!=NULL; temp=next)         // 1 mark
    {
        next = temp->next;                                    // 1 mark
        delete temp;                                          // 1 mark
   }
```

(COMP2011)[2016](s)final~=f0bidkdp^_68579.pdf downloaded by dgmwong from http://petergao.net/ustpastpaper/down.php?course=COMP2011&id=19 at 2020-10-28 05:40:07. Academic use within HKUST only.

```
}
```

(c) *[3 marks]* The member function bool `IntegerList::exists(int n)` will return true if the number, `n`, exists in the current `IntegerList` object; otherwise returns false.

```
bool IntegerList::exists(int n) const
{
   // Add your code here
   for (ll_node* temp = head;  temp!=NULL; temp=temp->next)   // 1 mark
   {
       if (temp->data == k)                                  // 1 mark
          return true;
   }
   return false;                                            // 1 mark

}
```

(d) *[4 marks]* The member function `void IntegerList::addItem(int n)` will first check if the integer number, `n`, exists in the current `IntegerList` object or not. If it does not exist, then add it to the **end** of the linked list; otherwise, do nothing.

```
void IntegerList::addItem(int n)
{
   // Add your code here
       if (!exists(n))                              // check exisits: 0.5 mark
       {
          ll_node* newNode = new ll_node;           // create a new node: 1 mark
          newNode->data = n;
          newNode->next = NULL;
          if (head == NULL)                      // add to an empty list: 1 mark
          {
              head = newNode;
          } else {
              ll_node* temp = head;          // go to the end of the list: 1 mark
              for (; temp->next!=NULL; temp=temp->next);
              temp->next = newNode;                    // add the node: 0.5 mark
          }
      }
```

(COMP2011)[2016](s)final~=f0bidkdp^_68579.pdf downloaded by dgmwong from http://petergao.net/ustpastpaper/down.php?course=COMP2011&id=19 at 2020-10-28 05:40:07. Academic use within HKUST only.

```
}
```

(f) *[4 marks]* The member function `void IntegerList::deleteItem(int n)` will remove the integer number, `n`, from the linked list.

```
void IntegerList::deleteItem(int n)
{
   // Add your code here

    if (head == NULL)                                 // empty list: 0.5 mark
       return;

    if (head->data == n) {                            // delete head: 1.5 mark
       ll_node* temp = head;
       head = head->next;
       delete temp;
       return;
    }

                       // search the remaining list nodes and delete: 2 marks
    ll_node* prev = head;
    ll_node* current = prev->next;
    for (; (prev->next != NULL) && (prev->next->data!=n); prev= prev->next,
current=current->next);
    prev->next = current->next;
    delete current;
```

(COMP2011)[2016](s)final~=f0bidkdp^_68579.pdf downloaded by dgmwong from http://petergao.net/ustpastpaper/down.php?course=COMP2011&id=19 at 2020-10-28 05:40:07. Academic use within HKUST only.

```
    }
```

(e) *[4 marks]* The member function `IntegerList IntegerList::union(const IntegerList& other)` will create and return a third `IntegerList` object containing all distinct elements from the current `IntegerList` object and the `other IntegerList` object, as the union of the two input lists.

```
IntegerList IntegerList::union(const IntegerList& other) const
{
   // Add your code here

                                   // any order of adding the items will be accepted
  IntegerList result;                                   // create a new list: 0.5 mark
                                          // adding items of this list: 1.5 marks
  for (ll_node* temp = head; temp!=NULL; temp=temp->next)
  {
     result.addItem(temp->data);
  }
                                          // adding items of the other list: 1.5 marks
  for (ll_node* temp = other.head; temp!=NULL; temp=temp->next)
  {
     result.addItem(temp->data);
  }
  return result;                                        // 0.5 marks

}
```

Part II. [5 marks] In an unordered list, one simple way to decrease the average search time is the move-to-front operation. When an element is accessed, it moves an element to the front of the list. This ensures that the most recently accessed items will be the quickest to find again.
Therefore, a new private member function, `void moveToFront(int)` will be added to the `IntegerList` class to improve the searching by moving the number `n` in the linked list to the front. If `n` is not in the list, do nothing.
For example, the original linked list is

9 -> 5 -> 3 -> 1 -> 2 -> 4

And after moving the item 3 in the linked list to the front, the linked list becomes

(COMP2011)[2016](s)final~=f0bidkdp^_68579.pdf downloaded by dgmwong from http://petergao.net/ustpastpaper/down.php?course=COMP2011&id=19 at 2020-10-28 05:40:07. Academic use within HKUST only.

3 -> 9 -> 5 -> 1 -> 2 -> 4

Implement the function bellow.

```
void IntegerList::moveToFront(int n)
{
   // Add your code here

       if ((head == NULL) || (head->next == NULL)||(head->data == n))
         return;
                                                       // 3 cases: empty list,
                                                       // only one node in the list,
                                                       // the number is the head node
                                                       //                  1.5 marks

       ll_node* prev= head;      // find the node and the previous node: 2 marks
       ll_node* temp = prev->next;
       for (; temp!=NULL; temp=temp->next, prev=prev->next)
       {
         if (temp->data == n)
           break;
       }
       if (temp != NULL)      // update the prev node and head pointer: 1.5 marks
       {
         prev->next = temp->next;
         temp->next = head;
         head = temp;
      }




}
```

(COMP2011)[2016](s)final~=f0bidkdp^_68579.pdf downloaded by dgmwong from http://petergao.net/ustpastpaper/down.php?course=COMP2011&id=19 at 2020-10-28 05:40:07. Academic use within HKUST only.

~ End of Paper ~

/* Rough work — DO NOT DETACH THIS PAGE */

(COMP2011)[2016](s)final~=f0bidkdp^_68579.pdf downloaded by dgmwong from http://petergao.net/ustpastpaper/down.php?course=COMP2011&id=19 at 2020-10-28 05:40:07. Academic use within HKUST only.

/* Rough work — You may detach this page */

(COMP2011)[2016](s)final~=f0bidkdp^_68579.pdf downloaded by dgmwong from http://petergao.net/ustpastpaper/down.php?course=COMP2011&id=19 at 2020-10-28 05:40:07. Academic use within HKUST only.

/* Rough work — You may detach this page */