COMP 2011 Final Exam - Fall 2014 - HKUST

Date:	Mon, December 15, 2014
Time allowed:	3 hours, 16:30–19:30
Instructions:	(1) This is a closed-book, closed-notes examination.
	(2) There are **12** questions on **26** pages.
	(3) There are also **4** pages of scrap paper, which you do not need to submit and therefore can be detached from the exam paper.
	(4) Answer **all** questions in the space after "**Answer:**", unless otherwise stated by the specific question.
	(5) You are recommended to answer all questions in black or blue ink, but pencils may also be used if you so wish.
	(6) All codes described in the questions are ANSI C++ compliant.
	(7) All your answer codes should also be ANSI C++ compliant.

Student Name	
Student ID	
Email Address	
Lecture & Lab Section	

For T.A. use only

Problem	Score
1	/ 5
2	/ 5
3	/ 5
4	/ 4
5	/ 4
6	/ 5
7	/ 8
8	/ 5
9	/ 15
10	/ 15
11	/ 14
12	/ 15
Total	/ 100

Problem 1 [**5 points**] True or false questions

Indicate whether the following statements are **true** or **false** by *circling* **T** or **F**, respectively. You get **1** point for each correct answer, −**0.5** for each wrong answer, and **0** point if you do not answer.

[✓ **T**] **F** (a) A compiler compliant with the ANSI C++ standard does not check if an array index is out of bound.

T [✓ **F**] (b) In the formal parameters of a function, only the leftmost ones may have default values.

[✓ **T**] **F** (c) No memory is allocated for both the formal parameters and the function definition of `area()` when the following statement is processed by a C++ compiler:

```
double area(double, double);
```

T [✓ **F**] (d) Consider the following variable definition:

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

Both `x` and `y` are pointer variables that point to locations for storing `int` values.

T [✓ **F**] (e) After the following program segment, the pointer variable `p` will point to `b`.

```
int a = 100, b = 200;
int* p = &a;
int* q = &b;
*p = *q;
```

Problem 2 **[5 points]** Multiple-choice questions

Circle the letter to the left of the correct answer for each question. There is only *one* correct answer per question. Each question is worth **1** point.

(a) Given the following definition of the function `func()`:

```
void func(int a, int b)
{
    cout << "hello" << endl;

    if (a > b)
        func(a-2, b);
    else if (a < b)
        func(a, b-1);
}
```

how many times does the call `func(10, 3)` print out the string `"hello"`?

(A) 4

(B) 5

(C) 6

(D) 7

(E) 8

(b) Which of the following is/are possible function header(s) of a working `swap()` function that swaps the values of two `int` variables?

(I) `void swap(int x, int y)`
(II) `void swap(int& x, int& y)`
(III) `void swap(int* x, int* y)`

(A) I only

(B) II only

(C) III only

(D) I and II only

(E) II and III only

(c) What is the output of the following program?

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

int main( )
{
    char s[20];
    int i;

    for (i = 0; i < 10; i++)
        *(s + i) = 'A' + i;
    *(s + i) = '\0';

    cout << s << endl;
    return 0;
}
```

(A) `abcdefghij`

(B) `0123456789`

(C) `AAAAAAAAAA`

(D) `ABCDEFGHIJ`

(E) no printable output

(d) Determine the output when the following program is run on a 32-bit machine.

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

int main( )
{
    const char* course = "Design and Analysis of Algorithms";
    cout << course[sizeof(course)] << endl;

    return 0;
}
```

(A) `g`

(B) `n`

(C) `0`

(D) `\0`

(E) runtime error called *segmentation fault* will occur

(e) Given the following two structures

```
struct Name
{
    const char* first_name;
    const char* last_name;
};

struct Record
{
    Name name;
    int age;
};
```

and the following statement

```
Record* namelist = new Record[10];
```

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

(I) `namelist[3].name.last_name = "Lee";`
(II) `namelist[3].name→last_name = "Lee";`
(III) `namelist[3]→name.last_name = "Lee";`
(IV) `namelist[3]→name→last_name = "Lee";`
(V) `namelist→name.last_name = "Lee";`

(A) I only
(B) IV only
(C) I and V only
(D) I, IV and V only
(E) none of them

Problem 3 [5 points] Array

We use an `int` array to represent a set (or collection) of integer values with the assumption that no two elements in the array have the same value. Suppose we have an array `set1` of size `size1` and another array `set2` of size `size2`, where both `size1` and `size2` are positive integers. The two sets represented by the arrays are said to be disjoint if they contain no common elements. Implement the following function for checking if the two sets represented by the two array parameters are disjoint:

```
bool disjoint(int set1[], int set2[], int size1, int size2);
```

It should return `true` if the sets are disjoint and `false` otherwise.

Answer:

```
bool disjoint(int set1[], int set2[], int size1, int size2)
{
    for (int i = 0; i < size1; ++i)
        for (int j = 0; j < size2; ++j)
            if (set1[i] == set2[j])
                return false;          // 2.5 POINTS for returning false
    return true;                       // 2.5 POINTS for returning true
}
```

Grading scheme:
- each kind of syntax error: -0.5 point, max 2 points

Problem 4 [**4 points**] Function overloading

What is the output when the following program is run?

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

int func(int& a, int& b)
{
    return (a + b);
}

int func(int* a, int* b)
{
    return (*a * *b);
}

double func(double a, double b)
{
    return (a - b);
}

int main(void)
{
    double x1 = 2.5;
    double y1 = 3.5;
    cout << func(x1, y1) << endl;

    int x2 = static_cast<int>(x1);
    int y2 = 10;
    cout << func(&x2, &y2) << endl;
    cout << func(x2, static_cast<double>(y2)) << endl;

    int* x3 = &x2;
    int* y3 = new int;
    *y3 = 5;
    cout << func(*x3, *y3) << endl;

    return 0;
}
```

Answer:

```
-1                                  // 1 POINT per number
20
-8
7
```

Problem 5 [**4 points**] Scope

What is the output of the following program?

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

int n = 1;

void print_plus_n(int x)
{
    cout << x + n << endl;
}

void increment_n()
{
    n = n + 2;
    print_plus_n(n);
}

int main(void)
{
    int n;
    n = 200;
    print_plus_n(7);
    n = 50;
    increment_n();
    cout << n << endl;

    return 0;
}
```

Answer:

```
8                                   // 1 POINT
6                                   // 1.5 POINTS
50                                  // 1.5 POINTS
```

Problem 6 **[5 points]** Linked list and recursion

Given a linked list of nodes whose type is defined as follows:

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

implement a *recursive* function,

```
void reverse_print(const Node* head);
```

to print the linked list in *reverse* order. The head of the linked list is pointed by the variable `head` of type `const Node*`. If the linked list is empty, nothing will be printed.

Answer:

```
void reverse_print(const Node* head)
{
    if (head == NULL)               // base case, 2 POINTS
        return;

    reverse_print(head->next);      // recursive case, 2 POINTS
    cout << head->data << " ";      // print node, 1 POINT
}
```

Grading scheme:
each kind of syntax error: -0.5 point, max 2 points

Problem 7 [8 points] Array and linked list

Write the function `array2sorted_ll()` to insert integers originally stored in an `int` array into a linked list of nodes of type `ll_inode` so that the integers are sorted in non-descending order (from small to large values) in the linked list. Below is a sample test program of the required function.

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

struct ll_inode
{
    int data;
    ll_inode* next;
};

/* Parameters:
 *    x: an integer array
 *    num_items: number of integers in x
 * Return: The head pointer of the resulting sorted linked list.
 * Remarks:
 *    - An empty array is represented by a NULL pointer with num_items = 0.
 *    - An empty linked list is represented by a NULL pointer.
 */
ll_inode* array2sorted_ll(const int x[], int num_items);

/* To print out the integers stored in a linked list */
void ll_print(const ll_inode* head)
{
    for (const ll_inode* p = head; p != NULL; p = p->next)
        cout << p->data << ' ';
    cout << endl;
}

int main( )
{
    int x[] = { 2, 8, 5, 6, 1, 4, 5, 2, 2 }; // there may be duplicates
    int num_items = sizeof(x)/sizeof(int);

    ll_inode* list = array2sorted_ll(x, num_items);
    ll_print(list);

    return 0;
}
```

When the program is run, its output is:

```
1 2 2 2 4 5 5 6 8
```

Answer:

```
// To create an ll_inode and initialize its integer data.
ll_inode* ll_create(int a)
{
    ll_inode* p = new ll_inode; p->data = a; p->next = NULL; return p;
}

ll_inode* array2sorted_ll(const int x[], int num_items)
{
    // Base case: empty list                           // 0.5 POINT
    if (x == NULL)
        return NULL;

    // STEP 1: Create the first ll_inode
    ll_inode* head = ll_create(x[0]);                  // 1 POINT for one node only

    // STEP 2: Add each subsequent inode in sorted order
    for (int j = 1; j < num_items; ++j)
    {
        ll_inode* new_inode = ll_create(x[j]);         // 1 POINT

        if (new_inode->data < head->data) // Prepend the new_inode to the front
        {                                              // 2 POINTS
            new_inode->next = head;
            head = new_inode;
        }
        else
        {
            ll_inode* prev = NULL;
            ll_inode* current = head;

            // Find the inserting position              // 2 POINTS
            while (current != NULL && new_inode->data > current->data)
            {
                prev = current;
                current = current->next;
            }

            // Actual insertion                         // 1 POINT
            new_inode->next = current;
            prev->next = new_inode;
        }
    }

    return head;                                        // 0.5 POINT
}
```

Grading scheme:
- each kind of syntax error: -0.5 point, max 2 points

Problem 8 [5 points] Class definition

Give the complete definition of a class called `Rectangle` which consists of *only one* constructor and one `const` member function `area()` such that the following code works as specified in the comments:

```
Rectangle r1;              // create object r1 with width = height = 10
Rectangle r2(4, 5);        // create object r2 with width = 4, height = 5
int r2_area = r2.area();   // store the area of r2 in r2_area
```

The definitions of the constructor and member function `area()` should be given separately *outside* the class definition. You may assume that the width and height of any object of `Rectangle` must be positive integers. Proper access control should be provided for the class members.

Answer:

```
class Rectangle
{
private:                                    // 0.5 POINT
    int width, height;                      // 0.5 POINT

public:                                     // 0.5 POINT
    Rectangle(int w = 10, int h = 10);      // 1 POINT
    int area() const;                       // 0.5 POINT
};

Rectangle::Rectangle(int w, int h)          // 1 POINT for whole constructor
{
    width = w;
    height = h;
}

int Rectangle::area() const                 // 1 POINT for whole function
{
    return width * height;
}
```

Grading scheme:
- each kind of syntax error: -0.5 point, max 2 points

Problem 9 [15 points] Stack with linked list implementation

A stack is an abstract data type (ADT) with last-in first-out (LIFO) policy. In this question we implement a stack with essentially unbounded capacity using a linked list. Each node in the linked list is a structure as defined below:

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

The head node of the linked list, which corresponds to the top of the stack, is pointed by the variable `top` of type `StackNode*`. For an empty stack, `top` has a `NULL` value. Recall that the push and pop operations of a stack only happen at its top.

The stack in this question supports the following operations:

```
void push(stackNode*& top, int value); // push an item into the stack
int pop(stackNode*& top);     // remove the top item from the stack and
                              // return its value (-1 if the stack is empty)
int size(const stackNode* top);        // return the number of items in the
                                       // stack (0 if the stack is empty)
void reverse(stackNode*& top);         // reverse the order of the stack
```

For simplicity, we assume that all values in the stack are positive integers.

(a) [5 points] Implement the `int pop(stackNode*& top)` function that removes the top item from the stack and returns its value. It returns `-1` if the stack is empty.

Answer:

```
int pop(stackNode*& top)
{
    if (top == NULL)                    // test empty stack, 1 POINT
        return -1;                      // return -1, 1 POINT
    else
    {
        stackNode* topNode = top;
        int topValue = top->value;
        top = top->next;                // update top pointer, 1 POINT
        delete topNode;                 // delete top item, 1 POINT
        topNode = NULL;                 // good practice, 0 POINT
        return topValue;                // return top value, 1 POINT
    }
}
```

Grading scheme:
- memory leak, i.e., lost the stack: -2 points

(b) [4 points] What is the output of the following program in which the `main()` function uses the `push`, `pop`, and `size` operations defined above?

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

int main(void)
{
    stackNode* stack = NULL;
    const int MAX = 4;

    for (int i = 0; i < 10; i++)
    {
        if (i % 4 == 0)
        {
            if (size(stack) != 0)
                cout << "pop: " << pop(stack) << endl;
        }
        else
        {
            if (size(stack) < MAX)
            {
                push(stack, i);
                cout << "push: " << i << endl;
            }
        }
    }

    return 0;
}
```

Answer:

```
push: 1                    // 0.5 POINT each
push: 2
push: 3
pop: 3
push: 5
push: 6
pop: 6
push: 9
```

(c) [6 points] Implement the `reverse(stackNode*& top)` function that reverses the order of the stack.

Hints/assumptions:

- You may assume that the `push()`, `pop()` and `size()` functions have been implemented and can be used if necessary.
- When additional memory space is used, you should make sure that there is no memory leak.

Answer:

```
void reverse(stackNode*& top)
{
    stackNode* reverseStack = NULL;
    while (size(top) > 0)                // iterate over all items, 2 POINTS
        push(reverseStack, pop(top));    // reverse order, 3 POINTS
    top = reverseStack;                  // reset stack top, 1 POINT
}
```

Grading scheme:
- any memory leak: -2 points

Problem 10 [15 points] Class definition

A calendar date is composed of the year, month, and day, and in addition, the weekday (i.e., day of the week). You are to implement a class called `Date` for a calendar date. Shown below is the definition of the `Date` class:

```
class Date
{
private:
    int year, month, day;
    char weekday[10];

public:
    Date();
      // default constructor (sets year, month, and day to 0 and
      // weekday to "")

    Date(const Date& another_date);
      // another constructor (creates a copy of another_date)

    Date(const char yyyymmdd[], const char wday[]);
      // another constructor (sets year, month, and day according to the
      // character string yyyymmdd and sets weekday to wday

    void set_date(const char yyyymmdd[], const char wday[]);
      // sets year, month, day, and weekday like the constructor above

    void print() const;                 // prints the date

    bool is_later_than(const Date& another_date) const;
      // returns true if the current object is later than another_date,
      // and false otherwise
};
```

The following is a `main()` function for manipulating `Date` objects, where `duplicate()` is a non-member function to be implemented:

```
int main(void)
{
    Date today("2014/12/15", "Monday");
    Date ten_days_ago;

    ten_days_ago.print();
    ten_days_ago.set_date("2014/12/05", "Friday");
    cout << "Ten days ago: "; ten_days_ago.print();
    cout << "Today: "; today.print();

    Date xmas2014("2014/12/25", "Thursday");
    Date new_date = duplicate(ten_days_ago);
    cout << "New date: "; new_date.print();

    if (new_date.is_later_than(xmas2014))
        cout << "New date is later than Christmas day" << endl;
    else
        cout << "New date is not later than Christmas day" << endl;

    return 0;
}
```

When the program is run, the following output is produced:

```
0-0-0 ()
Ten days ago: 2014-12-5 (Friday)
Today: 2014-12-15 (Monday)
New date: 2014-12-5 (Friday)
New date is not later than Christmas day
```

For simplicity, you may assume that all `Date` objects created are valid calendar dates.

The `char* strcpy(char* destination, const char* source)` function may be used to copy the C string pointed by `source` into the array pointed by `destination`. In addition, the `int atoi(const char* str)` function may be used to convert a C string pointed by `str` to an integer.

Grading scheme:
- each kind of syntax error: -0.5 point, max 2 points

(a) [3 points] Implement the following constructor:

```
Date(const Date& another_date);
```

Answer:

```
Date::Date(const Date& another_date)          // 0.5 POINT
{
    year = another_date.year;                 // 0.5 POINT
    month = another_date.month;               // 0.5 POINT
    day = another_date.day;                   // 0.5 POINT
    strcpy(weekday, another_date.weekday);    // 1 POINT
}
```

(b) [4 points] Implement the following member function:

```
void set_date(const char yyyymmdd[], const char wday[]);
```

You may assume that the first parameter is always a character string `"yyyy/mm/dd"` consisting of 10 characters, where `yyyy`, `mm`, and `dd` correspond to the year, month, and day, respectively.

Answer:

```
void Date::set_date(const char yyyymmdd[], const char wday[])
{                                          // 0.5 POINT for header
    char str[5];

    str[4] = '\0';                         // 1 POINT for year
    for (int i = 0; i < 4; ++i)
        str[i] = yyyymmdd[i];
    year = atoi(str);

    str[2] = '\0';                         // 1 POINT for month
    str[0] = yyyymmdd[5];
    str[1] = yyyymmdd[6];
    month = atoi(str);

    str[0] = yyyymmdd[8];                  // 1 POINT for day
    str[1] = yyyymmdd[9];
    day = atoi(str);

    strcpy(weekday, wday);                 // 0.5 POINT for weekday
}
    // -1 POINT for missing '\0'
```

(c) [2 points] Implement the following constructor using the `set_date()` member function:

```
Date(const char yyyymmdd[], const char wday[]);
```

Answer:

```
Date::Date(const char yyyymmdd[], const char wday[])
{                                          // 0.5 POINT for header
    set_date(yyyymmdd, wday);              // 1.5 POINTS
}
```

(d) [4 points] Implement the following member function:

```
bool is_later_than(const Date& another_date) const;
```

Answer:

```
bool Date::is_later_than(const Date& another_date) const
{
    if (year > another_date.year)
        return true;
    else if (year < another_date.year)
        return false;
    else if (month > another_date.month)
        return true;
    else if (month < another_date.month)
        return false;
    else if (day > another_date.day)
        return true;
    else
        return false;
                      // 2 POINTS for returning true correctly
                      // 2 POINTS for returning false correctly
}
```

(e) [2 points] Implement the following non-member function:

```
Date duplicate(const Date& date);
```

It creates and returns a new `Date` object with the values of its data members copied from those of the object passed to the function as a parameter.

Answer:

```
Date duplicate(const Date& date)          // 0.5 POINT
{
    Date new_date(date);                  // 1 POINT

    return new_date;                      // 0.5 POINT
}
```

Problem 11 [14 points] Array-based linked lists

Array-based linked lists are linked lists that are implemented using arrays, and the usual `next` pointer that indicates the following node is replaced by an array index. In the definition of the structure `ll_node` in the following program, notice that its `next` component is not a pointer variable but a `short int` which represents the index of the next item in the array-based linked list. Here we assume that the linked list will not have more items than what a (signed) `short int` can hold. If an `ll_node` does not link to any other `ll_node`, its `next` component will have a value of `-1`.

Now in the `main()` function, you are given two array-based linked lists of `ll_node`'s, `list1` and `list2`, both of which have their `int` data items sorted in non-descending order. They represent the following lists of sorted integers:

```
1 -> 2 -> 3 -> 4 -> 4
```

and

```
1 -> 1 -> 3 -> 5
```

You are required to implement the functions `ll_print()` and `ll_merge()` in the following program to merge the integers in the two sorted linked lists onto an array of sorted integers.

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

// Definition of each node in a linked list which is implemented by an array
struct ll_node
{
    int data;
    short next;
};

// General function to print out an int array with a title
void print_array(const char* title, int x[], int num_items)
{
    cout << title << " has " << num_items << " items: " << endl;
    for (int j = 0; j < num_items; j++)
        cout << x[j] << ' ';
    cout << endl << endl;
}

/* To print out the data in the array-based linked list in their linked order
 *
 * Parameters:
 *    list: an array of const ll_node's
 *    num_items: number of ll_node's in the list
 *    head: the array index of the starting node of the array-based linked list
 */
void ll_print(const ll_node list[], int num_items, int head);
```

```
/* To merge 2 sorted array-based linked lists of ll_node's and put the
 * resulting sorted integers in an int array
 *
 * Parameters:
 *    list1, list2: two sorted array-based linked lists of integers
 *    head1, head2: array index of the starting node of list1/list2 respectively
 *    sm_array: an int array to put the merged result of the 2 sorted lists
 */
void ll_merge(const ll_node list1[], int head1,
              const ll_node list2[], int head2, int sm_array[]);

int main(void)
{
    ll_node list1[] = { {4, 1}, {4, -1}, {3, 0}, {2, 2}, {1, 3} };
    ll_node list2[] = { {3, 2}, {1, 3}, {5, -1}, {1, 0} };
    int head1 = 4; // The index of the first sorted item in list1
    int head2 = 1; // The index of the first sorted item in list2
    int num_items1 = sizeof(list1)/sizeof(ll_node);
    int num_items2 = sizeof(list2)/sizeof(ll_node);
    int total_num_items = num_items1 + num_items2;

    cout << "List#1 has " << num_items1 << " items: " << endl;
    ll_print(list1, num_items1, head1);
    cout << "List#2 has " << num_items2 << " items: " << endl;
    ll_print(list2, num_items2, head2);

    // An array of integers of list1 and list2 in non-descending sorted order
    int* sm_array = new int[total_num_items];
    ll_merge(list1, head1, list2, head2, sm_array);
    print_array("The sorted array", sm_array, total_num_items);
    return 0;
}
```

The output of the program is shown below.

```
List#1 has 5 items:
1 2 3 4 4

List#2 has 4 items:
1 1 3 5

The sorted array has 9 items:
1 1 1 2 3 3 4 4 5
```

(a) [5 points] Print out the integer data in the array-based linked list `list` where `head` is the array index of the starting node in the array-based linked list.

```
void ll_print(const ll_node list[], int num_items, int head);
```

Answer:

```
void ll_print(const ll_node list[], int num_items, int head)
{
    for (int j = head; j != -1; j = list[j].next)        // 1, 1, 1.5 POINTS
        cout << list[j].data << ' ';                      // 1 POINT

    cout << endl << endl;                                 // 0.5 POINT
}
```

(b) [9 points] Given two lists `list1` and `list2` with their `int` data items sorted in non-descending order (i.e., from small values to large values with possible duplicates), this function merges their sorted data (integers) into the `int` array `sm_array` so that it contains all the integer data in the two linked lists in non-descending order.

```
/* To merge 2 sorted array-based linked lists of ll_node's and put the
 * resulting sorted integers in an int array
 *
 * Parameters:
 *    list1, list2: two sorted array-based linked lists of integers
 *    head1, head2: array index of the starting node of list1/list2 respectively
 *    sm_array: an int array to put the merged result of the 2 sorted lists
 */
void ll_merge(const ll_node list1[], int head1,
              const ll_node list2[], int head2, int sm_array[]);
```

Grading scheme:
- each kind of syntax error: -0.5 point, max 2 points
- part (b): grade by what can be done

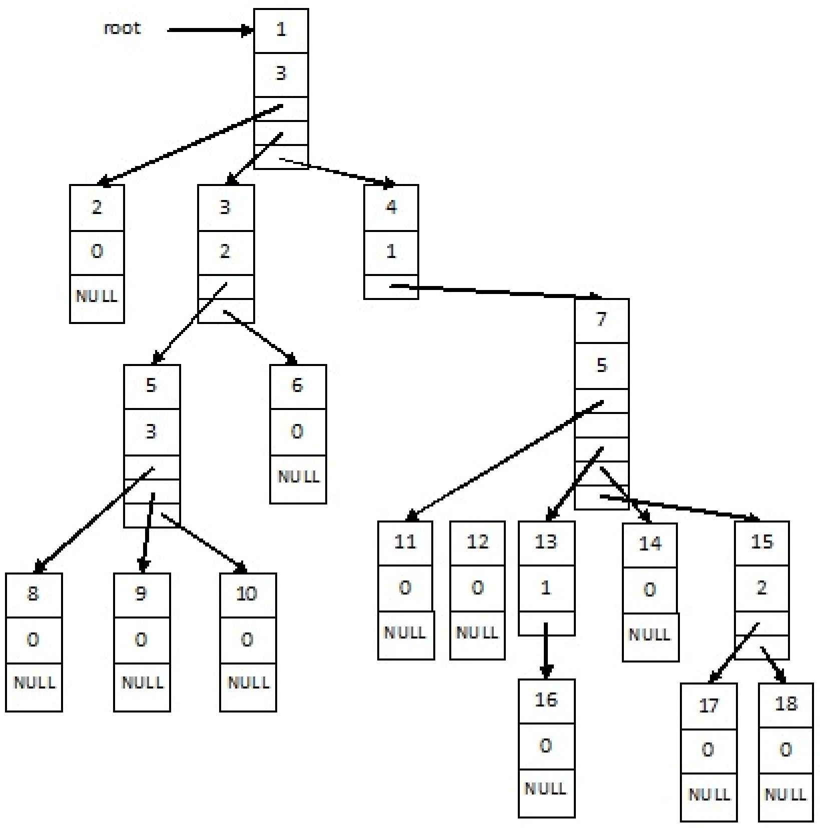
Answer:

```
// An auxiliary function to write ll_merge
void copy(const ll_node list[], int head, int sm_array[], int next_item)
{
    for (int j = head; j != -1; j = list[j].next)     // 1 POINT
        sm_array[next_item++] = list[j].data;         // 1 POINT
    // the 2 POINTS here are already included in the grading of ll_merge()
}

void ll_merge(const ll_node list1[], int head1,
              const ll_node list2[], int head2, int sm_array[])
{
    int i = 0;     // index for the next item to put in sm_array
    int j = head1; // index for the next integer in list1
    int k = head2; // index for the next integer in list2

    for ( ; j != -1 && k != -1; i++)        // While both lists are not exhausted
    {
        if (list1[j].data < list2[k].data)  // Insert values from list1
        {
            sm_array[i] = list1[j].data;    // 2 POINTS
            j = list1[j].next;              // 1 POINT
        }
        else                                // Insert values from list2
        {
            sm_array[i] = list2[k].data;    // 2 POINTS
            k = list2[k].next;              // 1 POINT
        }
    }

    // By now, one list should be exhausted.
    // Copy the remaining values of the non-empty list.
    if (j == -1)
        copy(list2, k, sm_array, i);        // 2 POINTS for the 1st copy
    else
        copy(list1, j, sm_array, i);        // 1 POINT for the 2nd copy
}
```

```
/***************** ALTERNATIVE SOLUTION USING RECURSION *******************/

void ll_merge_r(const ll_node list1[], int head1,
                const ll_node list2[], int head2,
                int sm_array[], int next_item)
{
    // Base case #1: list1 is empty; copy integers in list2 to sm_array
    if (head1 == -1)                                        // 1.5 POINTS
        copy(list2, head2, sm_array, next_item);

    // Base case #2: list2 is empty; copy integers in list2 to sm_array
    else if (head2 == -1)                                   // 1.5 POINTS
        copy(list1, head1, sm_array, next_item);

    // Case #3: list1's first data is smaller than list2's; append it to sm_array
    else if (list1[head1].data < list2[head2].data)     // 2 POINTS; 1 POINT
    {
        sm_array[next_item++] = list1[head1].data;
        ll_merge_r(list1, list1[head1].next, list2, head2, sm_array, next_item);
    }
    // Case #4: list2's first data is smaller than list1's; append it to sm_array
    else                                                    // 2 POINTS; 1 POINT
    {
        sm_array[next_item++] = list2[head2].data;
        ll_merge_r(list1, head1, list2, list2[head2].next, sm_array, next_item);
    }
}

// An alternative solution that use recursion
void ll_merge(const ll_node list1[], int head1,
              const ll_node list2[], int head2, int sm_array[])
{
    ll_merge_r(list1, head1, list2, head2, sm_array, 0);
}
```

Problem 12 [15 points] Dynamic tree

You are to implement some functions for the operations on a tree structure. Shown below is a sample tree with height 5 and a total of 18 nodes.



Each node in the tree is a structure defined as follows:

```
struct Node
{
    int data;
    int numOfChildren;
    Node** children;
};
```

The root node of the tree is pointed by variable `root` of type `Node*`.

In parts (a) and (b) of this question we assume that the nodes in a tree store distinct integer values.

(a) [4 points] Implement the following function:

```
bool searchTree(const Node* root, int value);
```

It searches the tree with its root node pointed by `root` to determine if the specified integer `value` exists in any node of the tree. It returns `true` if such a node is found and `false` otherwise. For example, given the sample tree, `search_Tree(root, 18)` returns `true` but `search_Tree(root, 101)` returns `false`.

Answer:

```
bool searchTree(const Node* root, int value)
{
    if (root == NULL)           // empty tree, or base case 1, 0.5 POINT
        return false;

    if (root->data == value)    // base case 2, 0.5 POINT
        return true;
                                // recursive search of sub-trees, 1 POINT
    for (int i = 0; i < root->numOfChildren; i++)
        if (searchTree(root->children[i], value))
            return true;        // return TRUE if found, 1 POINT

    return false;               // can't find in tree, 1 POINT
}
```

Grading scheme:
- tree is only partially searched: -2 points

(b) [4 points] Implement the following function:

```
int numOfNodes(const Node* root);
```

It counts and returns the number of nodes in the tree. For example, given the sample tree, it returns 18.

Answer:

```
int numOfNodes(const Node* root)
{
    if (root == NULL)   //empty tree, or base case, 0.5 POINT
        return 0;

    int num = 1;        // root node contribute to 1 in node count, 1 POINT
    for (int i = 0; i < root->numOfChildren; i++)   //for all sub-tree, 1 POINT
        num += numOfNodes(root->children[i]);       //add node count, 1 POINT

    return num;         // return node count, 0.5 POINT
}
```

(c) [7 points] Implement the following function:

```
void createTree(Node*& root, int max, int height);
```

It creates a *random* tree, with root node pointed by `root`, of the specified `height`. In addition, it has the following properties:

- Each node stores a random integer value within the range `1` to `100`.
- In each non-leaf node, `numOfChildren` is a random number within the range `1` to `max`.
- In each leaf node, `numOfChildren` is `0` and `children` is `NULL`.

For example, `createTree(root, 3, 5)` creates a tree of height 5 and each of its non-leaf nodes has 1 to 3 children.

Hints/assumptions:

- Multiple nodes may hold the same value.
- The height of the tree is assumed to be a positive integer (i.e., `height` $\geq$ 1).
- The `int rand(void)` function may be used to generate random non-negative integer values. You may assume that the necessary library has been included and the seed of the random generator has been properly set.

Answer:

```
void createTree(Node*& root, int max, int height)
{
    // Create the root node (coz height is at least 1)
    root = new Node;                                        // 0.5 POINT
    root->data = rand() % 100 + 1;                          // 0.5 POINT

    // Base case, 1 POINT
    if (height == 1)                                        // 0.5 POINT
    {
        root->numOfChildren = 0;                            // 0.5 POINT
        root->children = NULL;                              // 0.5 POINT
    }
    else // Recursive case
    {
        root->numOfChildren = rand() % max + 1;             // 0.5 POINT
        root->children = new Node*[root->numOfChildren];    // 1 POINT

        for (int i = 0; i < root->numOfChildren; i++)       // 1 POINT
            createTree(root->children[i], max, height-1);   // 2 POINTS
    }
}
```

Grading scheme:
- each kind of syntax error: -0.5 point, max 2 points