COMP 2011 Final Exam - Fall 2017 - HKUST

Date: December 12, 2017 (Tuesday)
Time Allowed: 3 hours, 12:30–3.30 pm
Instructions:

1. This is a closed-book, closed-notes examination.
2. There are **10** questions on **26** pages, including this cover page, an appendix page, and 2 blank pages at the end. You may tear out the blank pages for rough work.
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 the ANSI C++ version as taught in the class. Moreover, only the new C++11 features taught in class can be used. In particular, you are not allowed to use `auto`.
5. For programming questions, unless otherwise stated, you are **NOT** allowed to define additional structures, use global variables nor any library functions not mentioned in the questions.
6. Description of some C string and math functions can be found in the Appendix.

Student Name	Solution
Student ID	
Email Address	
Seat Number	

For T.A. Use Only

Problem	Score
1	/ 10
2	/ 5
3	/ 5
4	/ 8
5	/ 7
6	/ 4
7	/ 5
8	/ 16
9	/ 20
10	/ 20
Total	/ 100

Some Math and C String Library Functions Mentioned in the Questions or You May Need Them for Your Answers

```
/* strcpy() copies the string pointed to by s2 to the string pointed to by s1,
 * returning pointer s1.
 * The string pointed to by s1 should have enough memory to copy the
 * characters from the string pointed to by s2.
 */
char* strcpy(char* s1, const char* s2);
```

```
/* The strcmp() function compares the two given C strings s1 and s2, and
 * returns zero if the two strings are the same. If the two strings are
 * different, then it returns an integer less/greater than zero if for the first
 * characters that they differ, the character in s1 is smaller/larger than that
 * in s2 in terms of their ASCII codes.
*/
int strcmp(const char *s1, const char *s2);}
```

```
/* The pow() function returns the value of x raised to the power of y.
 */
double pow(double x, double y);
```

Problem 1 [10 points] True or False

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) The following code can be compiled without errors and runs to give the output "good".

```
int x = -1;
cout << ((++++x) ? "good":"bad") << endl;
```

T **F** (b) We may create an array of items of any type, including basic types (such as int, char, etc.), user-defined structs, pointers and references.

T **F** (c) The following code is illegal. That is, it is syntactically incorrect and cannot be compiled successfully.

```
const char* str = "hkust";
str[1] = 'K';
```

T **F** (d) The following code segment:

```
char str[] = "COMP2011";
cout << ++str;
```

can compile with no errors and run, and it prints out: `OMP2011`

T **F** (e) Assume that the size of `int` is 4 bytes and the size of pointer is 8 bytes in the following code.

```
int** ptr = reinterpret_cast<int**>(0x10);
cout << (ptr + 1) << endl;
```

The output of the above code segment is:

```
0x14
```

T **F** (f) The following function will create and return an identical copy of the C string in the char array `p`. Refer to the Appendix for the description of the function `strcpy`.

```
char* duplicate (const char p[])
{
    char* q = nullptr;
    strcpy(q, p);
    return q;
}
```

T **F** (g) The following code will result in memory leak.

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

struct Mystery { double data1; int* data2; };

void init(Mystery& m, double x, int y)
{
   m.data1 = x;
   m.data2 = new int; *(m.data2) = y;
}

int main()
{
   Mystery a; init(a, 1, 2);
   Mystery b = a;

   delete a.data2; a.data2 = nullptr;
   cout << b.data1 << " " << *(b.data2) << endl;
   return 0;
}
```

T **F** (h) Given the following definition of `Record` struct, **all** the following assignment statements are legal. That is, they can be compiled and run with no errors.

```
struct Record { int x; int y; };
Record a;
Record* p = &a;
```

- `p->x = 3;`
- `(*p).x = 3;`
- `a.y = 4;`
- `p->x = (&a)->y;`

T **F** (i) Any private or public class member may be used by any member function of the same class.

T **F** (j) A class called `Something` can include both of the following constructors in its class definition,

```
Something();
Something(int a = 0, float b = 0.0);
```

And a program may create a Something object as in the following example:

```
Something x;
```

Problem 2 [5 points] Loops

Implement the following function,

```
int sum(int from, int to, int diff);
```

that takes three integer arguments, namely, `from`, `to` and `diff`. The function returns the sum of a sequence of increasing or decreasing integers between `from` and `to` inclusively. The sequence of integers are obtained by either incrementing or decrementing each preceding number (except the first one) by `diff`, where `diff` is always a postive integer. You may further assume (and you don't need to check for these conditions) that the following conditions are always true when the function is called:

- `1 <= diff <= |to - from|`, and
- the arguments, `from` and `to`, are always different in their values.

For example,

```
cout << sum(2, 4, 1)   << endl;  // Output 9 because 2+3+4=9
cout << sum(7, 3, 1)   << endl;  // Output 25 because 7+6+5+4+3=25
cout << sum(7, 2, 2)   << endl;  // Output 15 because 7+5+3=15
cout << sum(-2, 7, 3)  << endl;  // Output 10 because -2+1+4+7=10
```

Answer:

Solution:

```
int sum(int from, int to, int diff)
{
   int total = 0;

   if (from < to)
   {

       for (int i = from; i <= to; i += diff)
          total += i;
   }
   else
   {
       for (int i = from; i >= to; i -= diff)
          total += i;
   }

   return total;
}
```

Problem 3 [5 points] Array and Pointer

What is the output when the following code is executed?

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

int main()
{
   int arr[10];
   for (int i = 0; i < 10; i++)
      arr[i] = i;

   int* a = arr + 9;
   while (*a > *arr)
   {
      a--;
      (*a)--;
      cout << *a << " ";
   }

   cout << endl;
   return 0;

}
```

Answer: 7 6 5 4 3 2 1 0

Problem 4 [4 points] Constructor and Destructor

What is the output when the following code is executed?

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

class Employee
{
  public:
    Employee() { id = salary = 0; cout << "Employee(" << id << ")" << endl; }
    ~Employee() { cout << "~Employee(" << id << ")" << endl; }
    void set(int sid, int s) {  id = sid; salary = s; }
  private:
    int id;
    int salary;
};

class Company
{
  public:
    Company(int n)
    {
        num_employees = n; employee = new Employee[n];
        cout << "Company(" << num_employees << ")" << endl;
        for (int i = 0; i < num_employees; i++)
            employee[i].set(100 + i, 10000);
    }
    ~Company() { delete [] employee; cout << "~Company()" << endl; }
  private:
    int num_employees;
    Employee *employee;
};

int main()
{
    Company c_ga_ga(2);
    return 0;
}
```

Answer:

Solution:

```
Employee(0)
Employee(0)
Company(2)
~Employee(101)
~Employee(100)
~Company()
```

Problem 5 [8 points] C String and Recursion

In this problem, you are asked to write 2 **recursive functions** about characters and C strings. NO LOOPS nor GOTO are allowed in your answers.

(a) [3 points] Write a recursive function with the following prototype

```
bool contains(const char* s, char c);
```

which returns true if the character `c` is contained in the C string `s`. For example,

```
contains("hkust", 'u');              // returns true
contains("hkust", 'a');              // returns false
contains("^@3&(85\**", '@');         // returns true
```

(b) [5 points] Write a recursive function with the following prototype

```
int num_distinct_char(const char* s)
```

to find out the number of distinct characters in the C string `s`. You may make use of part (a) to write your solution. For example,

```
num_distinct_char("eerie");          // returns 3
num_distinct_char("forever");        // returns 5
num_distinct_char("^@@@@^^");        // returns 2
```

Answer:

(a)

(b)

Solution:

```
/* Part (a) */
bool contains(const char* s, char c)
{
    if (*s == '\0')
        return false;

    return (*s == c) ? true : contains(s+1, c);
}



/* Part (b) */
int num_distinct_char(const char* s)
{
    if (*s == '\0')
        return 0;

    if  (contains(s+1, s[0]))
        return num_distinct_char(s+1);
    else
        return 1 + num_distinct_char(s+1);

/* Alternative answer:
    return (!contains(s+1, s[0])) + num_distinct_char(s+1);
*/
}
```

Problem 6 [7 points] Linked List

Implement the function:

```
node* merge(node* head1, node* head2);
```

which, given the head pointers of 2 linked lists of `node` structures, merges them together in place.

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

Here are some assumptions on the two input linked lists:

- Nodes in each of the linked lists are sorted in ascending order of their (integer) data.
- The data in each linked list are distinct.
- The data in one linked list are all different from the data in another linked list.

For example, if the values of the data in a linked list are 1, 9, 14, 27, and those in the other linked list are 12, 25, 36, the `merge` function should return a merged linked list whose nodes will contain the values: 1, 9, 12, 14, 25, 27, 36 in that order.

Here are more requirements for your solution of the `merge` function:

- It returns the head pointer of the merged list.
- If both input linked lists are empty, it returns an empty list.
- It should be an **in-place merge**. That is, you are **not allowed to create any new nodes during the merging process**, but manipulate the pointers with the existing nodes to get the required result.
- *Hint: Your solution can be either non-recursive or recursive.*

Answer:

Solution:

```
/* Recursive Solution */
node* merge(node* head1, node* head2)
{
    if (head1 == NULL) return head2;
    if (head2 == NULL) return head1;

    // pick either head1 or head2 to start, then recursion
    node* merged_head = nullptr;

    if (head1->data < head2->data)
    {
        merged_head = head1;
        merged_head->next = merge(head1->next, head2);
    }
    else
    {
        merged_head = head2;
        merged_head->next = merge(head1, head2->next);
    }

    return merged_head;
}
```

```
/* Iterative Solution */
node* merge(node* head1, node* head2)
{
    // Base cases
    if (head1 == NULL) return head2;
    if (head2 == NULL) return head1;

    // define working pointers
    node* merged_head = nullptr;
    node* p1 = head1; // p1 points to node to be inserted in list 1
    node* p2 = head2; // p2 points to node to be inserted in list 2
    node* p;          // p points to last node of the merged list

    // set merge_head
    if (head1->data < head2->data)
    {
        merged_head = head1;
        p1 = head1->next;
    }
    else
    {
        merged_head = head2;
        p2 = head2->next;
    }
    p = merged_head;

    // Merge: add nodes in the correct sequence by
    // weaving the merged list pointers
    while (p1 != nullptr && p2 != nullptr)
    {
        if (p1->data < p2->data)
        {
            p->next = p1;  // add node in list 1 to merged list
            p = p1;        // it's now last node in merged list
            p1 = p1->next; // move to next node in list 1
        }
        else
        {
            p->next = p2;
            p = p2;
            p2 = p2->next;
        }
    }

    // Handle the remaining list if reaching the end of one list
    p->next = (p1 == nullptr) ? p2 : p1;
    return merged_head;
}
```

Problem 7 [5 points] Lambda Expression

Write down the output for each of the following codes. If the code cannot be compiled, write "COMPILATION ERROR". If it compiles but may crash when it is run, write "RUNTIME ERROR".

(a)
```
int a = 4, b = 5, c = 6;
[](int x, int y, int z) { x = y; y = z; z = x; } (a, b, c);
cout << a << b << c << endl;
```

Answer: 456

(b)
```
int a = 4, b = 5, c = 6;
[](int& x, int& y, int& z) { x = y; y = z; z = x; } (a, b, c);
cout << a << b << c << endl;
```

Answer: 565

(c)
```
int a = 4, b = 5, c = 6;
[](int x, int y, int z) mutable { x = y; y = z; z = x; } (a, b, c);
cout << a << b << c << endl;
```

Answer: 456

(d)
```
int a = 4, b = 5, c = 6;
[=]( ) { a = b; b = c; c = a; } ( );
cout << a << b << c << endl;
```

Answer: COMPILATION ERROR

(e)
```
int a = 4, b = 5, c = 6;
[&]( ) { a = b; b = c; c = a; } ( );
cout << a << b << c << endl;
```

Answer: 565

Problem 8 [16 points] Classes

An album is a collection of audio recordings, for example, songs. In this question, you are asked to implement the `Album` class to store a list of songs together with their titles and durations as `Song` objects.

Below are an application program file "main.cpp" that creates an album (Album object) of songs (Song objects):

```
/* File: main.cpp */
#include "Album.h"

int main()
{
    // Initialize the data members, numSongs and maxDuration,
    // to 0 and 160 respectively
    Album WinterSelection(160);

    // Add a new Song object to an Album object.
    // If the total duration of the new song together with the current songs
    // in the Album object exceeds the Album object's maximum duration, return
    // false and do nothing. Otherwise, add the new Song object to the Album
    // object by updating its data members, and return true.
    Song songs[3];
    songs[0].set("Santa Claus is coming to town", 60);
    songs[1].set("Last Christmas", 105);
    songs[2].set("Snowman", 100);

    for (int i = 0; i <= 2; i++)
        cout << "Adding \"" << songs[i].getTitle() << "\": "
             << boolalpha << WinterSelection.addSong(songs[i]) << endl;

    // Output the titles of the Song objects in the Album object
    cout << "Songs in the album are:" << endl;
    WinterSelection.print();
    return 0;
}
```

and the corresponding output:

```
Adding "Santa Claus is coming to town": true
Adding "Last Christmas": false
Adding "Snowman": true
Songs in the album are:
Santa Claus is coming to town
Snowman
```

Here is the `Song` class definition file "Song.h":

```
#include <iostream>     /* File: Song.h */
#include <cstring>
using namespace std;

class Song
{
   public:
      Song();                          // Default constructor
      const char* getTitle() const;    // Return title
      int getDuration() const;         // Return duration
      void set(const char* t, int d); // Set the title & duration by t & d

   private:
      char title[100];                 // Title of the song
      int duration;                    // Duration of the song in seconds
};
```

(a) [3 points] Complete the class definition of the `Album` class in the file "Album.h" by writing the function prototypes of only the necessary member functions to support the operations in the main function in "main.cpp".
Note: Whenever a member function is an accessor, or a function argument should not be modified, you are required to specify it with `const`, otherwise marks will be deducted.

```
#include <iostream>     /* File: Album.h */
#include "Song.h"
using namespace std;
const int MAXSONGS = 10;

class Album
{
  public:
     /* Answer: */









  private:
    Song songs[MAXSONGS]; // Array of Song objects
    int numSongs;         // Number of Song objects stored in the songs
    int maxDuration;      // Maximum duration of the ablum in seconds
};
```

(b) [13 points] Implement the member functions of the `Album` class in a file called "Album.cpp".

Answer:

Solution:

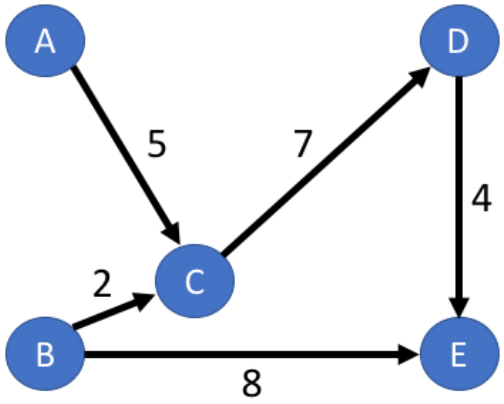
(a)

```
    Album(int);
    void print() const;        // Print all the songs' title
    bool addSong(const Song&); // Add a Song object to songs
```

(b)

```
#include "Album.h" /* File: Album.cpp */

// General constructor: initialize numSongs to 0 and maxDuration to d
Album::Album(int d)
{
   maxDuration = d;
   numSongs = 0;
}

// Accessor: output the titles of the songs in the album
void Album::print() const
{
   for (int i = 0; i < numSongs; i++)
      cout << songs[i].getTitle() << endl;
}

// Mutator: add a Song object, s, to the album
// If the total duration of s with the current songs exceeds the maximum
// duration of the album, return false and do nothing;
// otherwise, add s to the Album object by updating songs and numSongs,
// and return true.
bool Album::addSong(const Song& s)
{
   int total = 0;
   for (int i = 0; i < numSongs; i++)
      total += songs[i].getDuration();

   if ((total + s.getDuration()) <= maxDuration)
   {
      songs[numSongs++] = s;
      return true;
   }
   else
       return false;
}
```

Problem 9 [20 points] Dynamic Structure



The above picture (also called a graph) is an example of flights between 5 cities 'A'– 'E', which represent Athens, Beijing, Cairo, Dubai, and Edmonton, respectively. The arrows show the flights direction and the figure beside each arrow represents the cost of the flight in bitcoins!

Assumption: **NO** cycles among the flights. That is, there are no combination of flights to go from any city X back to city X.

The program in "flight.cpp" implements the flight information shown above with dynamic data structures defined in "flight.h". Read them carefully and implement the 3 missing functions with the given prototypes. For the other 3 utility functions, their actual codes are not important though you may probably guess them from the following output when the program is run:

```
There are the following flights:
Athens to Cairo : 5
Beijing to Cairo : 2
Beijing to Edmonton : 8
Cairo to Dubai : 7
Dubai to Edmonton : 4

Cost of some flights:
Athens to Cairo costs: 5
Athens to Edmonton costs: 16
Beijing to Dubai costs: 9
```

Important: Your answer should work for a general graph of any number of flights with no cycles, and not just the one in the above picture. You may use the C string function `strcmp`.

```
struct Flight             /* File: flight.h */
{
    const char* from;
    const char* to;
    int cost;
};

struct Flights
{
    int num_cities;
    const char** city;
    Flight*  flight;
    int num_flights;
};

#include <iostream>     /* File: flight.cpp */
#include "flight.h"
using namespace std;

// Print flight info in the format: departure-city "to" destination-city " : " cost
void print_one_flight(const Flight& flight);

// Print all the flights' info
void print_flights(const Flights& f);

// Initialize the info of one flight
void init_flight(Flight& flight, const char* c1, const char* c2, float cost);

/*^^^^^^^^^^^^^^^^^^^^^^^^  Functions to implement  ^^^^^^^^^^^^^^^^^^^^^^^^*/

/* Return true if there is a direct flight from city1 to city2, false
 * otherwise. Also find its cost if there is one, otherwise left it unchanged.
 * e.g., there is a direct flight from A to C (with cost = 5) but not A to D.
 */
bool direct_flight(const Flights& f, const char* city1, const char* city2, int& cost);

/* Return true if there is any direct or indirect flight from city1 to city2,
 * false otherwise. Also find its cost if there is one, otherwise left it unchanged.
 * E.g., there is a flight from A to E by taking 3 direct flights: A->C,C->D,D->E.
 * If there are more than 1 path (flight), report the cost of ANY ONE of them.
 * Hint: Make use of direct_flight() above and recursion.
 */
bool flight(const Flights& f, const char* city1, const char* city2, int& cost);

/* Delete all dynamic data allocated by the main function, and reset all
 * numerical parameters to zero, and all pointers to nullptr.
 */
void delete_flights(Flights& f);
```

```
/*!!!!!!!!!!  main function driver to test the required functions !!!!!!!!!!*/

int main()
{
    // Create strings for the city names
    char* A = new char [7]; strcpy(A, "Athens");
    char* B = new char [8]; strcpy(B, "Beijing");
    char* C = new char [6]; strcpy(C, "Cairo");
    char* D = new char [6]; strcpy(D, "Dubai");
    char* E = new char [10]; strcpy(E, "Edmonton");

    // Create the flights according to the picture above
    Flights f;
    f.city = new const char* [f.num_cities = 5];
    f.flight = new Flight [f.num_flights = 5];

    f.city[0] = A; f.city[1] = B; f.city[2] = C; f.city[3] = D; f.city[4] = E;
    init_flight(f.flight[0], A, C, 5);
    init_flight(f.flight[1], B, C, 2);
    init_flight(f.flight[2], B, E, 8);
    init_flight(f.flight[3], C, D, 7);
    init_flight(f.flight[4], D, E, 4);
    print_flights(f);

    // Find the cost of direct/indirect flights between some cities
    cout << endl << "Cost of some flights: " << endl;
    int cost = 0;
    if (direct_flight(f, A, C, cost)) // A->C exists; cost = 5
        cout << "Athens to Cairo costs: " << cost << endl;
    cost = 0;
    if (flight(f, A, B, cost)) // No flight, no print out
        cout << "Athens to Beijing costs: " << cost << endl;
    cost = 0;
    if (flight(f, A, E, cost)) // A->C->D->E; cost = 5+7+4 = 16
        cout << "Athens to Edmonton costs: " << cost << endl;
    cost = 0;
    if (flight(f, B, D, cost)) // B->C->D; cost = 2+7 = 9
        cout << "Beijing to Dubai costs: " << cost << endl;

    // Delete all dynamic memories allocated before AND reset all parameters
    delete_flights(f);
    return 0;
}
```

Answer:

(a) [6 points]

```
bool direct_flight(const Flights& f, const char* city1, const char* city2, int& cost)
```

(b) [9 points]

```
bool flight(const Flights& f, const char* city1, const char* city2, int& cost)
```

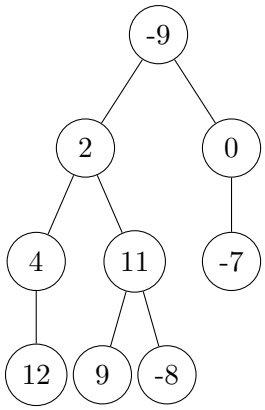
(c) [5 points] `void delete_flights(Flights& f)`

Solution:

```
bool direct_flight(const Flights& f, const char* city1, const char* city2, int& cost)
{
    for (int k = 0; k < f.num_flights; ++k)
        if (!strcmp(f.flight[k].from, city1) && !strcmp(f.flight[k].to, city2))
        {
            cost = f.flight[k].cost;
            return true;
        }

    return false;
}

bool flight(const Flights& f, const char* city1, const char* city2, int& cost)
{
    if (direct_flight(f, city1, city2, cost))
        return true;

    // Look for indirect flights
    for (int k = 0; k < f.num_flights; ++k)
        if (!strcmp(f.flight[k].from, city1))
        {
            if (flight(f, f.flight[k].to, city2, cost))
            {
                cost += f.flight[k].cost;
                return true;
            }
        }

    return false;
}

void delete_flights(Flights& f)
{
    for (int k = 0; k < f.num_cities; ++k)
        delete [] f.city[k];

    delete [] f.city;
    delete [] f.flight;
    f.city = nullptr;
    f.flight = nullptr;
    f.num_cities = 0;
    f.num_flights = 0;
}
```

Problem 10 [20 points] Level-order Traversal of a Binary Tree

This question is about binary trees in which each node has at most two children. For example, below is a binary tree with 9 nodes and it has a height = 4.



Each node in the binary tree is defined similarly as in our lecture notes as follows:

```
/* File: btree.h */
struct btnode
{
    int data;
    btnode* left;
    btnode* right;
};
```

A level-order traversal of a binary tree reads data from its nodes level by level from left to right, starting from the root. For example, a level-order traversal of the above binary tree will visit the nodes -9, 2, 0, 4, 11, -7, 12, 9, -8 in sequence. Implement the following 3 functions that should work for such binary trees of any size (and not just for the example tree above). Hint: Though both recursive and non-recursive solutions are allowed, consider to use recursion for part (a) and (b), and non-recursive solution for part (c).

(a) [5 points] `int height(const btnode* root)`
Return the height of a binary tree given its `root`. An empty tree has a height = 0.

(b) `void one_level_2_array(const btnode* root, int array[], int& index, int level)`
[9 points] Copy data from nodes in the given `level` of the binary tree with the given `root` from left to right into the given `array` starting at the given array `index`. Note:

- You may assume that `level` is always a valid tree level, $1 \leq level \leq height$.
- The root node is at level 1. In the above picture, nodes of 2 and 0 are at level 2, etc.
- You may also assume the array always has enough space to add data from the tree.
- `index` is the starting array index to insert the data from one level of the tree. E.g., calling `one_level_2_array(root, array, index, 3)` in the above tree when the initial `index = 3` will put 4, 11, and -7 into array[3], array[4] and array[5], respectively. After the function call, `index` becomes 6.

(c) [6 points] `int btree_2_array(const btnode* root, int array[])`
Copy data from the nodes in a binary tree with the given `root`, level by level from left to right to the `array`, and return the number of nodes (or data) in the tree. For the example tree, after calling the function, the contents of the array (starting from index 0) will be: -9, 2, 0, 4, 11, -7, 12, 9, -8, and the function returns 9. Assume that the array memory has already been allocated. Moreover, you may make use of the functions `height` and `one_level_2_array` in part (a) and (b) to implement this function.

For part (b) and (c), do nothing if the input tree is empty. Below is a sample testing program.

```
#include <iostream>
#include <cmath>
#include "btree.h"
using namespace std;

btnode* create_btree_node(int data)
{
    btnode* node = new btnode;
    node->data = data; node->left = node->right = nullptr;
    return node;
}

/*^^^^^^^^^^^^^^^^^^^^^^^^  Functions to implement  ^^^^^^^^^^^^^^^^^^^^^^^^*/
int height(const btnode* root);
void one_level_2_array(const btnode* root, int array[], int& index, int level);
int btree_2_array(const btnode* root, int array[]);

/*!!!!!!!!  main function driver to test the required 3 functions !!!!!!!!*/
int main()
{   /* Create the binary tree as shown in the above picture */
    btnode* root = create_btree_node(-9);
    root->left = create_btree_node(2);
    root->right = create_btree_node(0);
    root->left->left = create_btree_node(4);
    root->left->right = create_btree_node(11);
    root->right->left = create_btree_node(-7);
    root->left->left->left = create_btree_node(12);
    root->left->right->left = create_btree_node(9);
    root->left->right->right = create_btree_node(-8);

    // Create an empty array with enough memory to store the tree data
    int tree_height = height(root);
    int* array = new int [static_cast<int>(pow(2, tree_height)) - 1];

    // Copy all binary tree data to an array level by level
    int num_nodes = btree_2_array(root, array);

    // Print the tree/array data or information
    cout << "height of the binary tree = " << tree_height << endl; // height = 4
    cout << "size of the binary tree =  " << num_nodes << endl;    // size = 9
    cout << "content of the array: ";
    for (int i = 0; i < num_nodes; ++i)
        cout << array[i] << " ";     // -9 2 0 4 11 -7 12 9 -8
    cout << endl;
    /* Codes to release all dynamically allocated memory */
}
```

Answer:

(a) `int height(const btnode* root)`

(b) `void one_level_2_array(const btnode* root, int array[], int& index, int level)`

(c) `int btree_2_array(const btnode* root, int array[])`

Solution:

```
int height(const btnode* root)
{
    if (root == nullptr)
        return 0;

    int lheight = height(root->left);
    int rheight = height(root->right);
    return 1 + ((lheight > rheight) ? lheight : rheight);
}

void one_level_2_array(const btnode* root, int array[], int& index, int level)
{
    if (root == nullptr)
        return;

    if (level == 1)
    {
        array[index] = root->data;
        index++;
    }
    else if (level > 1)
    {
        one_level_2_array(root->left, array, index, level-1);
        one_level_2_array(root->right, array, index, level-1);
    }
}

int btree_2_array(const btnode* root, int array[])
{
    int tree_height = height(root);
    int index = 0;   // It also serves as the data count

    for (int level = 1; level <= tree_height; ++level)
        one_level_2_array(root, array, index, level);

    return index;
}
```

---------- END OF PAPER ----------