COMP 2011 Final Exam - Fall 2013 - HKUST

Date: Thu, December 12, 2013
Time allowed: 3 hours, 8:30–11:30
Instructions:
(1) This is a closed-book, closed-notes examination.
(2) There are **11** questions on **21** pages.
(3) There are also **3** 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 problem.
(5) You must answer all questions in black or blue ink (no pencils).
(6) All your answer codes should be ANSI C++ compliant, and no additional libraries nor global variables are allowed.

Student Name	Mr. Model Answer
Student ID	00000000
Email Address	dummy@comp2011.edu
Lecture & Lab Section	L123 LA1234

For T.A. use only

Problem	Score
1	/ 10
2	/ 10
3	/ 7
4	/ 8
5	/ 10
6	/ 6
7	/ 7
8	/ 10
9	/ 12
10	/ 20
11	/ 20
Total	/ 120

Problem 1 [10 points] True or false questions

Indicate whether the following statements are *true* or *false* by circling **T** or **F**. You get 2 points for each correct answer, −1 for each wrong answer, and 0 point if you do not answer.

T **F** (a) A function prototype has to specify the data type and name of all formal parameters.

T **F** (b) The following program runs and the output is 10.

```
int a = 10;
int *b = &a;
int *c = &b;
cout << **c << endl;
```

T **F** (c) Suppose that you wish to create a program that receives as input a set of numbers from the user (e.g., 1 2 3 4 5), and prints it in reverse order (e.g., 5 4 3 2 1). You are allowed to use either a *stack* with operations 'push' and 'pop', or a *queue* with operations 'enqueue' and 'dequeue'. Then, the best choice for solving this problem is the *stack*.

T **F** (d) In C++, the following two definitions of the variable *s* mean the same thing.

```
char *s = "Hello";
char s[ ] = "Hello";
```

T **F** (e) The output of the following program is 0.

```
int a[5] = {1,2};
cout << *(a+3) << end;
```

Problem 2 [10 points] Multiple choice questions

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 statements is true?

(a) `boolean` is a C++ keyword.
(b) `_0` is a valid C++ identifier. √
(c) Static objects obtain their memory from the heap.
(d) None of the above.

II. What is the value of the variable `i` after running the following code?

```
int i = 2;
while ((i *= 2) < 2000) ;
```

(a) 1024
(b) 2048 √
(c) 4096
(d) None of above

III. Assume that the struct A has been defined. Which of the following variable definition generates a compilation (syntax) error?

(a) `A a;`
(b) `A *aPtr = &a; // Assume a is of the type A`
(c) `A *a[6];`
(d) `A &aRef;` √

IV. Which of the following is not a correct function header of the `main` function?

(a) int main(void)
(b) int main(int argc, char** argv)
(c) int main(int argc, char*[ ] argv) √
(d) int main(int argc, char* argv[ ])

V. Consider the following code.

```
int *a = new int [10];
int *b = new int [10];
a = b;
```

The program causes:

(a) Compilation error
(b) Runtime error
(c) Memory leak √
(d) None of the above

Problem 3 [7 points] Value of π

The value of π can be determined using the following series:

$$\pi = 4 \times \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \frac{1}{13} - \ldots\right)$$

Complete the following C++ code fragment to approximate the value of π using the above formula including terms up to 1/99.

Answer:

```
double pi;

// ADD YOUR CODE HERE
double sum = 0;

for (int i = 0; i < 50; i++)
{
    if (i%2)
        sum -= 1.0/(2*i + 1);
    else
        sum += 1.0/(2*i + 1);
}

pi = 4.0 * sum;

// ADD YOUR CODE HERE — END
cout << "Value of pi = " << pi << endl;
```

Grading Scheme:

- a loop with correct number of iterations: 2 marks
- correct checking for odd/even terms: 2 mark
- correct double division for each term: 1 mark
- for each term, correctly determine when to add/subtract: 1 marks
- correct final result for pi: 1 mark
- Similar logic but use pow() function: -1 mark
- Repeated minor syntax errors: each -0.5 point, max -1 point.

Problem 4 [8 points] Pointers and pointer arithmetic

Consider the following C++ code segment.

```
int x = 5;
int y = 20;
int *ptra = &x;
int *ptrb = &y;
int *ptrc = ptra;
x = *ptrb / *ptrc;
ptrc += 1;
```

During program execution, the operating system stores the variables `x` and `y` at memory addresses 1000 and 2000 (in decimal) respectively. Fill in the following blanks with the correct values after the code fragment executes.

Answer:

`x` = 4

`*ptra` = 4

`*ptrb` = 20

`ptrc` = 1004

Grading Scheme: 2 marks for each correct answer

Problem 5 [10 points] Identifier declaration and const-ness

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

/* The following function set( ) is to be defined */
void set(..., int N);

void print(int x[ ], int N)
{
    for (int j = 0; j < N; ++j)
        cout << x[j] << ' ';
    cout << endl;
}

int main(void)
{
    const int N = 5;
    int* x = new int [N];

    set(x, N);
    print(x, N);
    return 0;
}
```

In the program above, the function `set` is yet to be defined. Consider each of the following definition of the function and for each of them, and answer (i) does the resulting complete program compile? (ii) if it compiles, does it run with no error? (iii) if it runs, write down the outputs.

Each part carries **2** points.

Grading Scheme:

- 2 points for each part
- for parts (a, b, d): 0.5 points for each correct sub-part
- for part (c): 1 points for each correct sub-part

Function Definition	Compile? (√ | ×)	Run? (√ | ×)	Outputs

(a)
`void set(int* x, int N)`
`{`
`  for (int j = 0; j < N; ++j)`
`  x[j] = 10 + j;`
`}`
	√	√	10 11 12 13 14

(b)
`void set(int* x, int N)`
`{`
`  for (int j = 0; j < N; ++j)`
`  x[j] = 20 + j;`
`  x = NULL;`
`}`
	√	√	20 21 22 23 24

(c)
`void set(int*& x, int N)`
`{`
`  for (int j = 0; j < N; ++j)`
`  x[j] = 30 + j;`
`  x = NULL;`
`}`
	√	×	

(d)
`void set(int* const x, int N)`
`{`
`  for (int j = 0; j < N; ++j)`
`  x[j] = 40 + j;`
`}`
	√	√	40 41 42 43 44

(e)
`void set(const int* x, int N)`
`{`
`  for (int j = 0; j < N; ++j)`
`  x[j] = 50 + j;`
`}`
	×		

Problem 6 [6 points] Mysterious function on linked lists

Consider the following mysterious function on linked lists:

```
struct node
{
    int data;                          // integer stored in a linked list node
    node* next;                        // Points to the next node
};

node* mystery(node *head, int k, int &i)
{
    if (head == NULL)
        return NULL;

    node* p = mystery(head->next, k, i);
    i = i+1;

    if (i == k)
        return head;

    return p;
}
```

(a) [4 points] Suppose we start with a linked list with the following contents:

$$1 \to 2 \to 3 \to 4 \to 5 \to 6$$

What is the output after the following program segment is executed?

```
int i = 0;
int k = 2;
node* q = mystery(head, k, i);
cout << q->data << endl;
```

Answer: 5

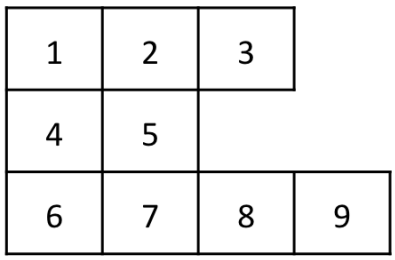
(b) [2 points] In a single sentence, what does the function '`mystery`' return?

Answer: The k^{th} to last node in the list

Grading Scheme:

- no partial points.
- +2 bonus points: if answer $(k - i)$th node from the end of the linked list.

Problem 7 [7 points] Dynamic 2D array with variable number of columns



The above figure shows an example of a table with a different number of columns in each row. Let's write a general C++ function `create_ragged_2d_array` to represent such kind of table by a dynamic 2D array by completing the function code below. Note that the above figure is just an example; your answer should support any number of rows and columns which are obtained from the formal parameters in the function header.

You may assume that the number of rows and columns is always greater than zero.

```
int** create_ragged_2d_array(const int num_columns[ ], int num_rows)
{                                        /* ADD YOUR ANSWER CODE HERE */

    int** array = new int* [num_rows];                    // Statement 1
    for (int j = 0; j < num_rows; ++j)                    // Statement 2
        array[j] = new int [num_columns[j]];              // Statement 3

    return array;                                         // Statement 4
}
```

Grading Scheme:

- Statement1: 2 points
- Statement2: 1 point
- Statement3: 3 points (1 for the form, 2 for correctness)
- Statement4: 1 point
- Repeated minor syntax errors: each -0.5 point, max -1 point.

Problem 8 [10 points] Class

Given the following class `Time` for a time display in 24 hour format:

```
class Time
{
    private:
        int hour;                                          // range 0 to 23
        int minute;                                        // range 0 to 59
        int second;                                        // range 0 to 59
    public:
        Time( );
        void advance(const Time&);
        void print( ) const
              { cout << hour << ":" << minute << ":" << second << endl; }
};
```

(a) [2 points] Implement the default constructor, which initializes a `Time` object to 0:0:0.

Answer:

```
Time::Time( )
{
    hour = 0;
    minute = 0;
    second = 0;
}
```

Grading Scheme:

- correct function header: 1 point
- correct function body: 1 point

(b) [8 points] Implement the member function `advance`, which moves a `Time` object's own time forward by adding to it the time of the input argument. For example, if an object's time is 0:40:20, and the time of the input argument is 4:40:40, result of calling `advance` by the object will advance its time to 5:21:00. Similarly, if the object's time is 20:40:20, its time becomes 1:21:00 after advancing by 4:40:40.

Answer:

```
void Time::advance(const Time& rhs)
{
    second += rhs.second;
    if (second >= 60)
    {
        second -= 60;
        minute++;
    }

    minute += rhs.minute;
    if (minute >= 60)
    {
        minute -= 60;
        hour++;
    }

    hour += rhs.hour;
    if (hour >= 24)
        hour -= 24;
}
```

Grading Scheme:

- correct function header: 1 point
- correctly access the data members: 1 point
- correct calculation of the hour: 2 points
- correct calculation of the minute: 2 points
- correct calculation of the second: 2 points
- Repeated minor syntax errors: each -0.5 point, max -1 point.

Problem 9 [12 points] Recursion and 2D arrays

Consider the following definitions:

```
enum Color {B, W, R, G, Y};
Color image[10][10];
```

Suppose that the 2D array '`image`' is used to represent a picture of 10×10 pixels, where the cell '`image[x][y]`' corresponds to the pixel (x, y). Each pixel is assigned one of the colors in the enum type '`Color`': black (B), white (W), red (R), green (G), or yellow (Y). You are required to implement the following function:

```
void fill_image(Color image[ ][10], int x, int y, Color old_color, Color new_color);
```

which takes as input the image, a certain pixel in the image represented by indices (x, y), an old color ('old_color'), and a new color ('new_color'). The function starts from (x, y) and, if the color of (x, y) is equal to 'old_color', it substitutes it with 'new_color'. It then proceeds recursively to all the pixels surrounding (x, y), and so on. The recursion must stop when either

- the color of the currently investigated pixel is *not* equal to 'old_color', or
- when it reaches an index out of the image array bounds.

For example, suppose that the original image is initialized as follows:

B	B	B	B	B	B	B	B	B	B
B	W	W	W	W	W	W	W	W	B
B	W	B	B	B	B	B	B	W	B
B	W	B	B	B	B	B	B	W	B
B	W	B	B	B	B	B	B	W	B
B	W	B	B	B	B	B	B	W	B
B	W	B	B	B	B	B	B	W	B
B	W	B	B	B	B	B	B	W	B
B	W	W	W	W	W	W	W	W	B
B	B	B	B	B	B	B	B	B	B

Then, if we call the function as:

```
fill_image(image, 3, 4, B, Y);
```

the content of the image becomes:

B	B	B	B	B	B	B	B	B	B
B	W	W	W	W	W	W	W	W	B
B	W	Y	Y	Y	Y	Y	Y	W	B
B	W	Y	Y	Y	Y	Y	Y	W	B
B	W	Y	Y	Y	Y	Y	Y	W	B
B	W	Y	Y	Y	Y	Y	Y	W	B
B	W	Y	Y	Y	Y	Y	Y	W	B
B	W	Y	Y	Y	Y	Y	Y	W	B
B	W	W	W	W	W	W	W	W	B
B	B	B	B	B	B	B	B	B	B

Answer:

```
void Fill(Color image[][10], int x, int y, Color old_color, Color new_color)
{
    // ADD YOUR ANSWER CODE HERE‘
    if(x < 0 || y < 0 || x >= 10 || y >= 10)
        return;

    if(image[x][y] == old_color)
    {
        image[x][y] = new_color;

        Fill(image, x-1, y, old_color, new_color);
        Fill(image, x+1, y, old_color, new_color);
        Fill(image, x, y-1, old_color, new_color);
        Fill(image, x, y+1, old_color, new_color);
    }
}
```

Grading Scheme:

- base case: 3 points
- correct if-condition: 1 point
- correctly assign new color to the investigating pixel: 2 points
- 4 correct recursive calls: 6 points (each 1.5 points)
- if use 8 recursive calls instead: 6 points (each 6/8 points and round up to the nearest 0.5 point)
- Repeated minor syntax errors: each -0.5 point, max -1 point.

Problem 10 [20 points] Binary tree.

Complete the following 3 functions in the given program on binary trees:

(a) [7 points] `create_btree`: It creates a binary tree with *depth* levels of nodes, each storing an integer. The stored values follows a pattern: if a non-leaf node stores the value n, then its left child will store the value $3 \times n - 1$ and its right child will store the value $3 \times n + 1$. For a leaf node, it stores the value required by its parent node. The value stored at the root is obtained from the first argument.

(b) [7 points] `flip_btree`: It flips a binary tree against an imaginary central axis through the root so that the resulting tree is a mirror image of the original one. That is, the left sub-tree becomes the right sub-tree and vice versa, and this is done recursively.

(c) [6 points] `print_btree_nth_level`: It prints the values stored in the nodes at the nth level of a binary tree from left to right, one on each output line. We count levels of a binary tree from 1.

You may assume that the depth and printing level are always positive integers greater than zero, and the printing level is not greater than the depth of the tree.

Grading Scheme: Penalties

- part (a):
  - wrong depth: -1 point
  - wrong argument in recursive call: -0.5 point for each one
- part (b):
  - base case can be: if (root-¿right == NULL)
- part (c):
  - wrong level: -1 point
  - wrong level in recursive call: -0.5 point
  - wrong order, i.e., print right node before left node: -1 point
- Repeated minor syntax errors: each -0.5 point, max -1 point.

Below is a screen capture from a sample run of the program.

```
Enter root number, depth of tree, and printing level: 2 4 3

The original tree
                        67
                22
                        65
        7
                        61
                20
                        59
2
                        49
                16
                        47
        5
                        43
                14
                        41


The flipped tree
                        41
                14
                        43
        5
                        47
                16
                        49
2
                        59
                20
                        61
        7
                        65
                22
                        67


Values at the 3th level
22
20
16
14
```

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

struct btnode                                   // A node in a binary tree
{
    int data;
    btnode* left;                             // Left sub-tree or called left child
    btnode* right;                           // Right sub-tree or called right child
};

// Create one single binary tree node that stores the given data
// The code below is the same as in our lecture notes and is reproduced here.
btnode* create_btnode(int data)
{
    btnode* node = new btnode;
    node->data = data;
    node->left = node->right = NULL;
    return node;
}

// Print data stored on a binary tree side-wise
void print_btree(const btnode* const tree, int depth = 0)
{
    if (tree == NULL)                                          // Base case
        return;

    print_btree(tree->right, depth+1);                // Recursion: right subtree
    for (int j = 0; j < depth; j++)                      // Print the node data
        cout << "  ";
    cout << tree->data << endl;
    print_btree(tree->left, depth+1);                  // Recursion: left subtree
}
```

```
btnode* create_btree(int root_num, int depth);
void flip_btree(btnode* const root);
void print_btree_nth_level(const btnode* const root, int level);

int main(void)
{
    int root_num;                                   // Number stored at the root
    int depth;                                  // Number of levels of tree nodes
    int level;                                 // The level of nodes to be printed

    cout << "Enter root number, depth of tree, and printing level:  ";
    cin >> root_num >> depth >> level;

    // Create a tree with a depth given by the variable depth
    btnode* root = create_btree(root_num, depth);
    cout << endl << endl << "The original tree" << endl ;
    print_btree(root);                                  // Print the resulting tree

    // Flip the tree so that it becomes the mirror image of its original one.
    flip_btree(root);
    cout << endl << endl << "The flipped tree" << endl ;
    print_btree(root);                                  // Print the resulting tree

    // Print data stored on the nth depth level from left to right
    cout << endl << endl << "Values at the " << level << "th level" << endl ;
    print_btree_nth_level(root, level);
    return 0;
}

/* (a)
 * Aim: To create a binary tree with btnode's. Each node stores an integer. For
 *      each non-leaf node, if it stores the value n, then its left child will
 *      store the value 3*n-1 and its right child will store the value 3*n+1.
 * Parameters:
 *      root_num : number to be stored at the root of a binary tree.
 *      depth : number of levels of tree nodes.
 */
btnode* create_btree(int root_num, int depth)
{                                  /* ADD YOUR ANSWER CODE HERE */

    if (depth == 1)                                     // Base case: 2 points
        return create_btnode(root_num);

    btnode* root = create_btnode(root_num);              // New node: 1 point
    root->left = create_btree(3*root_num - 1, depth-1);  // Recursion: 1.5 points
    root->right = create_btree(3*root_num + 1, depth-1);// Recursion: 1.5 points
    return root;                                        // Correct return: 1 point
}
```

```
/* (b)
 * Aim: To flip a binary tree against the central axis through the root so that
 *      the resulting tree is a mirror image of the original one.
 * Parameters:
 *      root : pointer to the root of the binary tree.
 */
void flip_btree(btnode* const root)
{                                   /* ADD YOUR ANSWER CODE HERE */

    if (root->left == NULL)                          // Base case: 2 points
        return;

    flip_btree(root->left);                          // Recursion: 1.5 points
    flip_btree(root->right);                         // Recursion: 1.5 points

    btnode* temp = root->left;                       // Correct swap: 2 points
    root->left = root->right;
    root->right = temp;
}
```

```
/* (c)
 * Aim: To print the values stored in the nth level nodes of a binary tree
 *      from left to right, one value per line.
 * Parameters:
 *      root : pointer to the root of the binary tree.
 *      level : the level of nodes to print. (Note that levels start from 1.)
 */
void print_btree_nth_level(const btnode* const root, int level)
{                                   /* ADD YOUR ANSWER CODE HERE */

    if (level == 1)                                  // Base case: 3 points
    {
        cout << root->data << endl;
        return;
    }

    print_btree_nth_level(root->left, level-1);      // Recursion: 1.5 points
    print_btree_nth_level(root->right, level-1);     // Recursion: 1.5 points
}
```

Problem 11 [20 points] Linked lists

```
struct node
{
    int data;                          // integer stored in a linked list node
    node* next;                                    // Points to the next node
};
```

Write 3 functions to manipulate a linked list of nodes whose C++ struct definition is given above.

(a) [6 points] Complete the following function by writing just the necessary C++ code to dynamically create the linked list that represents the following:

$$5 \rightarrow 7 \rightarrow 2 \rightarrow NULL\ .$$

```
/*
 * Aim: Create the particular linked list with 3 nodes with the following
 *       contents: $5 \rightarrow 7 \rightarrow 2$.
 * Return: a pointer to the beginning of the linked list.
 */
node* create_linkedlist(void)
{
    node* head;
    // ADD YOUR ANSWER CODE HERE

    head = new node;

    head->value = 5;
    head->next = new node;

    head->next->value = 7;
    head->next->next = new node;

    head->next->next->value = 2;
    head->next->next->next = NULL;

    return head;
}
```

Grading Scheme:

- 1 point for each of the 8 lines of code
- round to the nearest 0.5 point
- repeated minor syntax errors: each -0.5 point, max -1 point.

(b) [6 points] Implement the following function `delete_node` which takes as input a pointer p to a node in the *middle* of a linked list and delete it from the linked list. For example, if the original list looks like:

$$1 \to 2 \to 3 \to 4 \to 5 \to NULL \ ,$$

and p points to the node with data 3, after calling the function, the resulting linked list should look like the following:

$$1 \to 2 \to 4 \to 5 \to NULL \ .$$

You may assume that there is always *at least one* node preceding $*p$, and *at least one* node succeeding $*p$ in the list.

[Hint: (i) You don't have the pointer to the preceding node of $*p$, and (ii) we only require that the resulting linked list "*looks as if*" the node $*p$ is deleted.]

```
/*
 * Aim: To delete a node in the middle of a linked list without
 *      using a previous pointer.
 * Parameter:
 *      p: a pointer to the node to be deleted
 */
void delete_node(node* p)
{
    node* q = p→next;                    // Create a pointer to the next node

    // Copy info from the next node to the current node
    p→data = p→next→data;
    p→next = p→next→next;                          // Or, *p = *(p->next);

    delete q;                                       // Delete the next node
}
```

Grading Scheme:

- correct copy of data+next pointer: 3 points
- correct setup and deletion of the required node: 3 points
- repeated minor syntax errors: each -0.5 point, max -1 point.

(c) [8 points] Implement the function `reverse_linklist` which reverses a linked list in-place — without using any extra data structures — and in one pass. For example, Given

$$1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow NULL$$

after calling the function, it becomes

$$5 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 1 \rightarrow NULL\ .$$

For simplicity, you may assume the linked list has at least 2 or more nodes.

```
/*
 * Aim: To reverse a linked list in place in one pass without creating any
 *      additional data structures.
 */
void reverse_linkedlist(node*& head)
{
    node* prev = NULL;
    node* current = head;
    node* next;

    // ADD YOUR ANSWER CODE HERE
    while (current != NULL)
    {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }

    head = prev;
}
```

Grading Scheme:

- successfully traverse through the linked list: 3 points
- correct pointer manipulations to reverse the linked list: 3 points
- correct *head* at the end: 2 points
- penalties:
  - any memory leak or dangling pointer, -3 points
  - repeated minor syntax errors: each -0.5 point, max -1 point.