COMP 2011 Final Exam - Spring 2014 - HKUST

Date: May 26, 2014 (Monday)
Time allowed: 3 hours, 12:30pm–3:30pm
Instructions:

1. This is a closed-book, closed-notes examination.
2. There are **11** questions on **27** pages (including this cover page).
3. There are also **3** pages of scrap paper, which you do not need to submit and therefore may 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 programming codes in your answers must be written in ANSI C++.
7. For programming questions, you are **NOT** allowed to define additional helper functions or structures, nor global variables unless otherwise stated. You also **are not allowed** to use any library functions not mentioned in the questions.

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

For T.A.

Use Only

Problem	Score
1	/ 15
2	/ 6
3	/ 8
4	/ 7
5	/ 6
6	/ 6
7	/ 6
8	/ 18
9	/ 16
10	/ 12
11	/ 25
Total	/ 125

Problem 1 [**15 points**] True or false questions

Indicate whether the following statements are *true* or *false* by circling **T** or **F**. You get 1.5 points for each correct answer, −0.5 for each wrong answer, and 0 point if you do not answer.

[✓ **T**] **F** (a) If `a`, `b`, and `c` are all `int` variables with values greater than 1, then the following 2 expressions:

`1/(a*b*c)` and `1/a/b/static_cast<float>(c)`

give the same numerical result (regardless of its type).

[✓ **T**] **F** (b) The statement

`p->center = 10.0;`

can be equivalently written as: `(*p).center = 10.0;`

[✓ **T**] **F** (c) If `a` is an array identifier, then the expression

`a[5]`

is equivalent to: `*(a+5)`

T [✓ **F**] (d) The following code will give compilation error(s).

```
int b = 5;
int* const a = &b;
*a = 3;
```

[✓ **T**] **F** (e) A C++ struct can be defined inside a function.

T [✓ **F**] (f) The public members of a class can *only* be accessed by its member functions.

T [✓ **F**] (g) The default access control of members (data or function) in a class definition is public.

T [✓ **F**] (h) A C++ class may have one or more constructors and one or more destructors.

T [✓ **F**] (i) If an object `x` of C++ class `MyClass` is created statically in a function as follows:

`MyClass x;`

At the end of the function, one has to call `MyClass`'s destructor to destruct `x`, otherwise, the stack will only release `x`'s memory without calling its destructor.

[✓ **T**] **F** (j) Continuing from the last question, in order to be able to create an array of MyClass objects (for example, `MyClass x[10];`), the program must have a default constructor (which takes no argument) for `MyClass`.

Problem 2 **[6 points]** If Statement

Given the following program

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

int main(void)
{
    int n;
    cout << "Enter an integer:  "; cin >> n;

    if (n < 10)
        cout << "less than 10" << endl;
    else if (n > 5)
        cout << "greater than 5" << endl;
    else
        cout << "not interesting" << endl;

    return 0;
}
```

(a) [1.5 points] What is the output if the user enters the integer 0?

Answer: **less than 10**

(b) [1.5 points] What is the output if the user enters the integer 7?

Answer: **less than 10**

(c) [1.5 points] What is the output if the user enters the integer 15?

Answer: **greater than 5**

(d) [1.5 points] What values of `n` will give the output "not interesting"?

Answer: **none**

Problem 3 [8 points] Sorting

You are given the following `swap` function.

```
void swap(int& a, int& b) { int temp = a; a = b; b = temp; }
```

(a) [3 points] Use the `swap` function to implement the following `sort3` function in no more than <u>3 statements</u>.

```
// Sort 3 integers in ascending order using several calls of swap
void sort3(int& a, int& b, int& c)
{
    if (a > b) swap(a, b);
    if (a > c) swap(a, c);
    if (b > c) swap(b, c);
}
```

Grading Scheme: 1 point for getting the smallest/greatest one in order

(b) [5 points] Use <u>only</u> the `sort3` function to implement the following `sort5` function in no more than <u>4 lines</u> of code.

```
// Sort 5 integers in ascending order using several calls of sort3
void sort5(int& a, int& b, int& c, int& d, int& e);
{
    sort3(a, b, c);
    sort3(a, b, d);
    sort3(a, b, e);                 // Now, a and b must be the smallest and a <= b
    sort3(c, d, e);
}

/* There can be other solutions as well. Below is one alternative. */
void sort5(int& a, int& b, int& c, int& d, int& e);
{
    sort3(a, b, c);
    sort3(b, c, d);
    sort3(c, d, e);                 // Now, d and e must be the largest and d <= e
    sort3(a, b, c);
}
```

Grading Scheme: 1—2 point for getting the 1—2 smallest/greatest one in order

Problem 4 [7 points] Array and Recursion

Given an array of integers, write a RECURSIVE function that returns the minimum integer in the array. The prototype of the function is:

```
int array_min(const int x[ ], int size);
```

where `size` is the number of elements in the array `x` and you may assume that `size` is always greater than 0.

*** No marks will be given for a non-recursive solution.

Answer:

```
// Assumption: size is always greater than zero
int array_min(const int x[ ], int size)
{
    // ADD YOUR CODE HERE
   if (size == 1)
        return x[0];

    int y = array_min(x, size-1);
    return (y < x[size-1]) ? y : x[size-1];

}
```

Grading Scheme:

- syntax errors: @0.5, max 1.0 point
- base case: 2 points
- correct use of recursion: 3 points
- correct returned value: 2 points

Problem 5 [6 points] Parameter Passing Methods

What is the output of the following program?

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

int a = 2;
int b = 3;

int F(int& a, int b)
{
    a = 8;
    return a+b;
}

int G(int x, int* y)
{
    int a = b;
    *y = a;
    return x+a;
}

int main(void)
{
    int x = 2014;
    int y = F(a, b);
    int z = G(a, &x);

    cout << " x = " << x << endl;
    cout << " y = " << y << endl;
    cout << " z = " << z << endl;
    return 0;
}
```

```
x = 3
y = 11
z = 11
```

Answer: ________________________________

Grading Scheme: 2 points for each line of output

Problem 6 [6 points] Data Types

Given the following identifier definitions:

```
char c = 'A';
char* p = &c;
char** p2 = &p;
int* v = reinterpret_cast<int*>(p);
```

write down the type (e.g., int, void, etc.) of the following expressions. If the expression is syntactically incorrect (i.e., it cannot be compiled without errors), write the word "ILLEGAL". The first one is given to you as an example.

Expression	**Type**
e.g., `p`	`char*`
(a) [1.5 points] `&p2`	char***
(b) [1.5 points] `*p2`	char*
(c) [1.5 points] `p2+1`	char**
(d) [1.5 points] `v[0]`	int

Problem 7 [6 points] Pointer Operations

The effect of the combination of the derefencing operator and the incrementing operator on pointers can be confusing sometimes. Determine the outputs of the following program.

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

int main(void)
{
    int x[ ] = { 10, 20, 30, 40, 50 };
    int k;
    int* p;

    for (p = x, k = 0; k < 2; ++k)
        cout << *(p++) << " ";                      // Answers for Part (a)
    cout << endl;

    for (p = x, k = 0; k < 2; ++k)
        cout << ++(*p) << " ";                      // Answers for Part (b)
    cout << endl;

    for (p = x, k = 0; k < 2; ++k)
        cout << *(++p) << " ";                      // Answers for Part (c)
    cout << endl;

    return 0;
}
```

(a) [2 points]

Answer: 10 20

(b) [2 points]

Answer: 11 12

(c) [2 points]

Answer: 20 30

Problem 8 [18 points] A Pixel Class

Digital pictures are composed of picture elements called 'pixels'. Each pixel is a small square element consisting of 3 basic light color components: red (R), green (G) and blue (B). Each of these RGB component has an integral value in the range of 0–255 (inclusive). A pixel with (R, G, B) = (0, 0, 0) represents the black color, whereas the pixel with (R, G, B) = (255, 255, 255) represents the white color.

In this problem, you are required to define a C++ class called `Pixel`. It should have

- 3 private data members: red, green, and blue. All of them should have the type of *unsigned int* and their values must be inside the range of 0–255.
- 1 public constructor which takes 3 integer arguments to initialize the red, green, and blue components in that order.
- 2 other public member functions:
  - `display`:
    this function takes no argument, returns nothing, and does not modify any data members. When called, it shows the values of the the R, G, B components in that order. For example, if the 3 components have the values of 10, 20, and 30 respectively, then the output will be:
    ```
    (R, G, B) = (10, 20, 30)
    ```
  - `brighten_up`:
    this function takes a positive floating point number of the type *double* called '*factor*'. The values of the R, G, B components will be updated by multiplying the factor. The results should be rounded. That is, if the product is 15.2 then the result should be 15; if the product is 109.8 then the result should be 110. If the factor is greater than 1.0, then the pixel will become brighter, whereas if the factor is smaller than 1.0, then it will become darker.

Note: If the result of any pixel value after some operation becomes smaller than 0, set it to 0; if it becomes greater than 255, set it to 255.

(a) [6 points] Write a complete C++ definition of the class Pixel, which is assumed to be saved in a file called “pixel.h”.

Answer:

```
/* File: pixel.h */

class Pixel
{
  private:
    unsigned int red; ;                                  // 0.5 point
    unsigned int green;                                  // 0.5 point
    unsigned int blue;                                   // 0.5 point

  public:
    Pixel(int r, int g, int b);                          // 1.5 points
    void display(void) const;                            // 1.5 points
    void brighten_up(double factor);                     // 1.5 points
};
```

Grading Scheme:

- syntax errors: @0.5, max 2.0 point for the whole question
- for the functions: -0.5 point for e.g., missing “const”, wrong type
- no double penalty for errors in both “pixel.h” and “pixel.cpp”.

(b) [12 points] Write the implementation of all the member functions of the class Pixel. They are assumed to be saved in a separate file called "pixel.cpp".

Answer:

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

Pixel::Pixel(int r, int g, int b)                        // Correct header: 1 point
{
    red = (r < 0) ? 0 : ( (r > 255) ? 255 : r );                  // Total 2 points:
    green = (g < 0) ? 0 : ( (g > 255) ? 255 : g );       // 0.5 point for 1 component
    blue = (b < 0) ? 0 : ( (b > 255) ? 255 : b );
}  // Check the range: Add 2 bonus points

void Pixel::display(void) const                          // Correct header: 1 point
{
    cout << "(R, G, B) = (" << red << ", " << green << ", " << blue << ")"
         << endl;                                                       // 2 points
}

/* Solution 1: brighten_up */
void Pixel::brighten_up(double factor)                   // Correct header: 1 point
{
    // 3 points: 1 for rounding
    Pixel x(red*factor + 0.5, green*factor + 0.5, blue*factor + 0.5);
    red = x.red;                                             // 2 points altogether
    green = x.green;
    blue = x.blue;
}
/* Solution 2: brighten_up */
void Pixel::brighten_up(double factor)                   // Correct header: 1 point
{
    int r = red*factor + 0.5;                                 // Rounding: 1 point
    int g = green*factor + 0.5;
    int b = blue*factor + 0.5;
    red = (r < 0) ? 0 : ( (r > 255) ? 255 : r );           // Check range: 2 points
    green = (g < 0) ? 0 : ( (g > 255) ? 255 : g );
    blue = (b < 0) ? 0 : ( (b > 255) ? 255 : b );    // Completely correct: 2 points
}
```

Problem 9 **[16 points]** Linked lists

```
#include "ll_cnode.h"

/* ADD FUNCTION DEFINITIONS HERE */
const ll_cnode* ll_max(const ll_cnode* head);
bool same_lls(const ll_cnode* head1, const ll_cnode* head2);
void ll_remove_even_nodes(ll_cnode* head);

int main(void)
{
    char s1[100], s2[100];
    cout << "Enter 2 char strings to compare:  ";
    cin >> s1 >> s2;

    // ll_create( ) converts an input string to a char linked list
    ll_cnode* x1 = ll_create(s1);
    ll_cnode* x2 = ll_create(s2);

    // ll_print( ) prints the content of the linked list
    cout << "x1:  length = " << ll_length(x1) << "\tdata = "; ll_print(x1);
    cout << "x2:  length = " << ll_length(x2) << "\tdata = "; ll_print(x2);

    // Find the largest character of x1 and x2
    const ll_cnode* max_node;                    // max_node points to the node with max value

    max_node = ll_max(x1);
    if (max_node) cout << "max char in x1 = " << max_node->data << endl;

    max_node = ll_max(x2);
    if (max_node) cout << "max char in x2 = " << max_node->data << endl;

    // Check if x1 and x2, which are represented by linked lists, are the same
    cout << endl << "The 2 input strings are the same:  ";
    cout << boolalpha << same_lls(x1, x2) << endl;

    // Remove the even nodes from x1 and x2
    ll_remove_even_nodes(x1);
    ll_remove_even_nodes(x2);

    cout << endl << "After removing even nodes ..." << endl;
    cout << "x1:  length = " << ll_length(x1) << "\tdata = "; ll_print(x1);
    cout << "x2:  length = " << ll_length(x2) << "\tdata = "; ll_print(x2);

    return 0;
}
```

Implement the 3 missing functions in the above program, namely *ll_max*, *same_lls*, and *ll_remove_even_nodes*. You may write additional helper functions if you find them necessary.

Below is a sample run of the program with the input "china switzerland":

```
Enter 2 char strings to compare: china switzerland
x1: length = 5 data = china
x2: length = 11 data = switzerland
max char in x1 = n
max char in x2 = z

The 2 input strings are the same: false

After removing even nodes ...
x1: length = 3 data = cia
x2: length = 6 data = sizrad
```

Grading Scheme: syntax errors: @0.5, max 2.0 point for the whole question

(a) [4 points]

```
/* Aim: To find the node that contains the largest value among all the nodes
 *       in the char linked list.
 * Input: head of the given char linked list
 * Return: a point to the node which has the max char value; NULL if the list
 *          is empty.
 * Remark: [1] the characters are compared simply by their ASCII codes.
 *          [2] The linked list cannot be modified.
 */
const ll_cnode* ll_max(const ll_cnode* head)
{     // ADD YOUR ANSWER CODE HERE
    const ll_cnode* maxnode = head;                          // 0.5 point

    // 1.5 points: 0.5 point for each of the 3 parts of "for heading"
    for (const ll_cnode* p = head; p; p = p->next)
    {
        if (p->data > maxnode->data)                         // 1.5 points
            maxnode = p;
    }

    return maxnode;                                          // 0.5 point
}
```

(b) [6 points]

```
/*
 * Aim: To check if two given char linked lists contains exactly the same
 *        characters (both in terms of the number of characters and their values)
 *        in exactly the same order.
 * Inputs: the head of 2 char linked lists
 * Return: true or false
 * Remarks: The 2 linked lists cannot be modified.
 */
bool same_lls(const ll_cnode* head1, const ll_cnode* head2)
{
    // ADD YOUR ANSWER CODE HERE
    if (!head1 && !head2)                                    // Case #1: 1 point
        return true;
    else if (head1 && head2)            // Case #2: total 4 points: condition 0.5
    {
        if (head1->data != head2->data)                            // 1 point
            return false;
        else
            return same_lls(head1->next, head2->next);         // 2.5 points
    }
    else
        return false;                                        // Case #3: 1 point
}
```

(c) [6 points]

```
/*
 * Aim: To remove the even nodes from a char linked list. That is, if the list
 *        originally has 6 nodes, you are required to remove the 2nd, 4th, and
 *        6th nodes. The resulting linked list then will only have 3 nodes. In
 *        general, if the linked list originally has 2N nodes, after the function,
 *        it contains only N nodes; if originally it has (2N+1) nodes, after the
 *        function, it only has (N+1) nodes.
 * Input: head of the given char linked list
 * Return: nil
 * Remark: [1] The memory of the removed even nodes MUST be released using the
 *                   delete operator.
 *             [2] There will be no change to the list if it has fewer than 2 nodes.
 */
void ll_remove_even_nodes(ll_cnode* head)
{
    // ADD YOUR ANSWER CODE HERE
    if (!head)                                              // Base case: 1 point
        return;

    ll_cnode* p = head->next;                               // 0.5 point

    if (p)      // The 2nd node is not empty => need to be removed; 0.5 point
    {
        head->next = p->next;                               // 1.5 points
        delete p;                                           // 1.5 points
        ll_remove_even_nodes(head->next);                   // 1 point
    }
}
```

Grading Scheme:

- return on NULL head: 1 point
- correctly set up the loop: 2 point
- correctly skip the even nodes: 2 point
- correct delete call: 1 point

Problem 10 [12 points] Conversion between Linked list and Stack

A character string is reversed using a character linked list and a character stack using the program on the next page as follows.

STEP 1 : Read a character string from the user into a character array.

STEP 2 : Create a character linked list to store the string using the function,

```
ll_cnode* ll_create(const char [ ])
```

taught in class.

STEP 3 : Copy the string *from left to right* from the char linked list to a char stack using the following new function,

```
void ll_2_stack(const ll_cnode* s, char_stack& a)
```

so that the leftmost (or the first) character of the string will be put to the bottom of the stack, and the rightmost (or the last) character of the string will be put to the top of the stack.

STEP 4 : Finally, the characters on the stack are popped out one by one, which are used to create a new character linked list using the new function,

```
ll_cnode* stack_2_ll(char_stack& a)
```

Implement the 2 missing functions, namely *ll_2_stack* and *stack_2_ll*.

Below is a sample run of the program with the input “emit”:

```
Enter a char string to reverse: emit
s2: length = 4 data = emit
No. of data currently on the stack: 4 Top item: t
time
```

```
#include "ll_cnode.h"
#include "char-stack.h"

/* ADD FUNCTION DEFINITIONS HERE */
void ll_2_stack(const ll_cnode* s, char_stack& a);
ll_cnode* stack_2_ll(char_stack& a);

void print_stack_info(const char_stack& s)
{
    cout << "No. of data currently on the stack: " << s.size( ) << "\t";
    if ( !s.empty( ) ) cout << "Top item: " << s.top( ) << endl;
}

int main(void)
{
    char s[100];
    cout << "Enter a char string to reverse: ";
    cin >> s;

    // STEP1: First put the string s on a linked list
    ll_cnode* s2 = ll_create(s);
    cout << "s2: length = " << ll_length(s2) << "\tdata = "; ll_print(s2);

    // STEP2: Now the characters of s2 are copied to a stack
    char_stack a;
    ll_2_stack(s2, a);
    print_stack_info(a);

    // STEP3: Characters on the stack are popped onto a new linked list
    ll_cnode* rev_s2 = stack_2_ll(a);
    ll_print(rev_s2);

    return 0;
}
```

(a) [5 points]

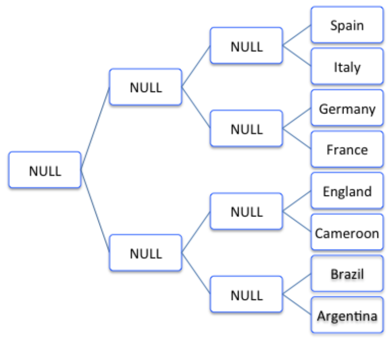
```
/*
 * Aim: To put a char string that is stored on a char linked list onto a char
 *        stack so that at the end, the first char will be stored at the bottom
 *        of the stack, and the last char at the top of the stack.
 * Input: head of the given char linked list (pass-by-value), and a char stack
 *          that is passed by reference.
 * Return: nil
 * Remark: [1] The given char stack is assumed to be empty and will be
 *                  modified by filling up with the characters in the given linked
 *                  list.
 *             [2] The linked list cannot be modified.
 */
void ll_2_stack(const ll_cnode* s, char_stack& a)
{
    // ADD YOUR ANSWER CODE HERE
    if (s)                                                    // Condition: 1 point
    {
        a.push(s->data);                               // Add data by push: 2 points
        ll_2_stack(s->next, a);                          // Recursion: 2 points
    }
}
```

(b) [7 points]

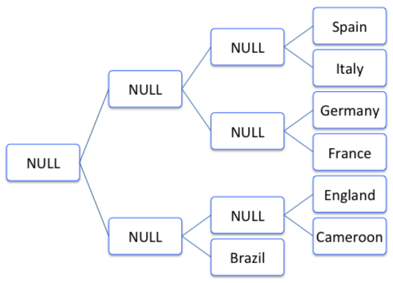
```
/*
 * Aim: To create a new char linked list to store a char string that is
 *        originally stored on a char stack. The char at the top of the stack
 *        should become the first char on the linked list, and last char at the
 *        bottom of the stack will become the last char on the linked list.
 * Input: The char stack that is passed by reference.
 * Return: The ll_cnode pointer pointing the head of the new char linked list.
 * Remark: [1] If the given char stack is empty, the returned value should
 *                  represent an empty linked list.
 *             [2] You may modify the given char stack.
 */
// Recursive solution
ll_cnode* stack_2_ll(char_stack& a)
{
    if (a.empty())                                        // Base case: 1 point
        return NULL;

    ll_cnode* head = ll_create(a.top());   // Create a ll_cnode for an item: 2 points
    a.pop();                                       // Get to the next item: 2 points
    head->next = stack_2_ll(a);                          // Recursion: 2 point
    return head;                            // If wrong return value: -0.5 point
}
```

```
// Non-recursive solution
ll_cnode* stack_2_ll(char_stack& a)
{
    if (a.empty())                                          // Base case: 1 point
        return NULL;

    ll_cnode* head = new ll_cnode;                   // Create new ll_cnode: 1 point
    head→data = a.top();              // Initialize the data pointed by head: 1 point
    a.pop();                                       // Get to the next item: 2 points

    ll_cnode* p = head;                               // p is the working pointer
    for (; !a.empty(); a.pop())                      // Correct loop: 2 points total
    {
        p→next = new ll_cnode;
        p = p→next;
        p→data = a.top();
    }
    p→next = NULL;                   // The last ll_cnode should point to NOTHING
    return head;                             // If wrong return value: -0.5 point
}
```

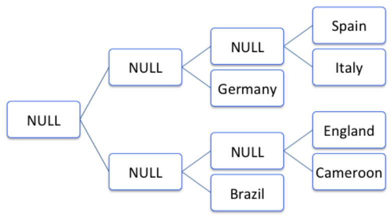
Problem 11 [25 points] Binary Tree and World Cup 2046



(a) final 8



(b) after Brazil wins



(c) after Germany wins

Let's use a binary tree with 4 levels (starting from the zero-th level) to represent the matches among the final 8 teams in the World Cup 2046 as shown in Fig. 1a. The binary tree nicely represents how the teams are paired up. The eight leaf nodes on the 3rd level give the 8 teams. If Brazil wins over Argentina, the resulting tree is described by Fig. 1b. And if Germany further wins over France, the tree will look like the one in Fig. 1c. Notice that, each leaf node has a char pointer pointing to the name of the (winning) team, while such char pointer in each internal node is NULL. After a match is over and the winner is known, the winner will 'move' upwards, and the leaves of the 2 teams in the match are removed as shown in Fig. 1b and Fig. 1c. The names of the 8 teams and the definition of the binary tree struct for this question are given by the file "worldcup.h" as shown below.

```
/* File: worldcup.h */
const char* team[ ] =                        // Names of the final 8 teams are stored here
{
    "Argentina",
    "Brazil",
    "Cameroon",
    "England",
    "France",
    "Germany",
    "Italy",
    "Spain"
};

struct btnode                                 // Definition of the binary tree node
{
    const char* winner;
    btnode* left;
    btnode* right;
};
```

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

void print_record(const btnode* record, int level = 0);
btnode* create_1_team(const char* team);
btnode* group_teams(btnode* left_group, btnode* right_group);

int main(void)
{
    // STEP1: Create 4 pairs of teams
    btnode* pair1 = group_teams(create_1_team(team[0]), create_1_team(team[1]));
    btnode* pair2 = group_teams(create_1_team(team[2]), create_1_team(team[3]));
    btnode* pair3 = group_teams(create_1_team(team[4]), create_1_team(team[5]));
    btnode* pair4 = group_teams(create_1_team(team[6]), create_1_team(team[7]));

    // STEP2: Put 2 pairs of teams in a group and 2 groups together in a tree
    btnode* quad1 = group_teams(pair1, pair2);
    btnode* quad2 = group_teams(pair3, pair4);
    btnode* record = group_teams(quad1, quad2);

    cout << "After putting the 8 teams onto the binary tree ..." << endl;
    print_record(record);

    cout << endl << "Brazil wins ..." << endl;
    /* (d) ADD STATEMENTS HERE TO IMPLEMENT THE WINNING OF BRAZIL
     *      AND THE RESULTING TREE
     */
    print_record(record);

    cout << endl << "Germany wins ..." << endl;
    /* (e) ADD STATEMENTS HERE TO IMPLEMENT THE WINNING OF
     *      GERMANY AND THE RESULTING TREE
     */
    print_record(record);

    return 0;
}
```

Implement the 3 missing functions in the program in the file "worldcup.cpp", namely *print_record*, *create_1_team*, and *group_teams*. The program first creates the binary tree representing Fig. 1a, then the tree in Fig. 1b and Fig. 1c.

Below is the output of the program.

```
After putting the 8 teams onto the binary tree ...
                        Spain
                        Italy
                        Germany
                        France
                        England
                        Cameroon
                        Brazil
                        Argentina

Brazil wins ...
                        Spain
                        Italy
                        Germany
                        France
                        England
                        Cameroon
                Brazil

Germany wins ...
                        Spain
                        Italy
                Germany
                        England
                        Cameroon
                Brazil
```

General Grading Scheme: syntax errors: @0.5, max 2.0 point for the whole question

(a) [5 points]

```
/* Aim: To create a binary tree node dynamically and initialize its winner
 *       with the given team.
 * Input: Name of the team to initialize the newly created tree node.
 * Return: Pointer of the newly created tree node.
 */
btnode* create_1_team(const char* team)
{
    // ADD YOUR ANSWER CODE HERE
    btnode* node = new btnode;
    node->winner = team;
    node->left = NULL;
    node->right = NULL;
    return node;
}
```

Grading Scheme: 1 point for each line of code

(b) [5 points]

```
/* Aim: To create a binary tree given a left sub-tree and a right sub-tree.
 *       Each sub-tree represents a group of matches between soccer teams.
 * Input: Roots of the left and right sub-trees.
 * Return: Pointer of the newly created root whose left pointer points to the
 *           left_group and whose right pointer points to the right_group.
 */
btnode* group_teams(btnode* left_group, btnode* right_group)
{
    // ADD YOUR ANSWER CODE HERE
    btnode* node = new btnode;
    node->winner = NULL;
    node->left = left_group;
    node->right = right_group;
    return node;
}
```

Grading Scheme: 1 point for each line of code

(c) [5 points]

```
/*
 * Aim: To print a binary tree nicely as shown in Fig. 1. In the printout, the
 *        tree is turned 90 degrees, so that the root (at the 0-th level) is
 *        shown in the leftmost column. Before printing the contents of the next
 *        level, a tab space is added so as to show the different levels of
 *        nodes. Since the internal nodes contain NULL char pointer for the
 *        winners, nothing are shown for them. Only the content of the leaf nodes
 *        are invisible in the printout.
 * Inputs: Head of a binary sub-tree and its level with respective to the
 *           whole tree
 * Return: nil
 */
void print_record(const btnode* record, int level = 0)
{
    // ADD YOUR ANSWER CODE HERE
    if (!record)                                          // Base case: 1 point
        return;

    if (record->winner)
    {
        for (int j = 0; j < level; j++)                   // Correct tabbing: 1 point
            cout << '\t';
        cout << record->winner << endl;                   // Output winner: 1 point
    }
    else
    {
        print_record(record->right, level+1);             // 1 point
        print_record(record->left, level+1);              // 1 point
    }
}
```

(d) [5 points] Add the C++ statements to hard-code the modifications needed in the binary tree with the root at `btnode* record` to represent the resulting tree after Brazil wins over Argentina as shown in Fig. 1b.

Answer:

```
record→left→left→winner = team[1];
delete record→left→left→left;
delete record→left→left→right;
record→left→left→left = NULL;
record→left→left→right = NULL;
```

Grading Scheme: 1 point for each line of code

(e) [5 points] Add the C++ statements to hard-code the modifications further needed in the binary tree to represent the resulting tree after Germany wins over France as shown in Fig. 1c.

Answer:

```
record→right→left→winner = team[5];
delete record→right→left→left;
delete record→right→left→right;
record→right→left→left = NULL;
record→right→left→right = NULL;
```

Grading Scheme: 1 point for each line of code

Appendix I: Definition of the ll_cnode struct

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

struct ll_cnode
{
    char data;                                        // contains useful information
    ll_cnode* next;                                   // the link to the next node
};

const char NULL_CHAR = '\0';
ll_cnode* ll_create(char);
ll_cnode* ll_create(const char [ ]);
int ll_length(const ll_cnode*);
void ll_print(const ll_cnode*);
ll_cnode* ll_search(ll_cnode*, char c);
void ll_insert(ll_cnode*&, char, unsigned);
void ll_delete(ll_cnode*&, char);
void ll_delete_all(ll_cnode*&);
```

Appendix II: Definition of the char-stack class

```
#include <iostream>                                        /* File: char-stack.h */
using namespace std;
const int BUFFER_SIZE = 1024;

class char_stack
{
  private:
    char data[BUFFER_SIZE];                             // Use an array to store data
    int top_index;                                     // Starts from 0; -1 when empty

  public:
    // CONSTRUCTOR member functions
    char_stack(void);                                            // Default constructor

    // ACCESSOR member functions: const => won't modify data members
    bool empty(void) const;                               // Check if the stack is empty
    bool full(void) const;                                  // Check if the stack is full
    int size(void) const;                      // Give the number of data currently stored
    char top(void) const;                              // Retrieve the value of the top item

    // MUTATOR member functions
    void push(char);                              // Add a new item to the top of the stack
    void pop(void);                                  // Remove the top item from the stack
};
```