Homework Tutorial 4: Software Design – TDN, GDN, and Refinement

Tianyi Yang

csci3100@cse.cuhk.edu.hk

March 24, 2021

Outline

1. Software Design Principle and Approach
2. Architecture Design and Design Notation
3. Program Design Technique (I)
4. Program Design Technique (II)
5. Software Design: Recursive and Non-Recursive Modules

1. Software Design Principle and Approach

Source: 2020 / Homework 4 / Question 1

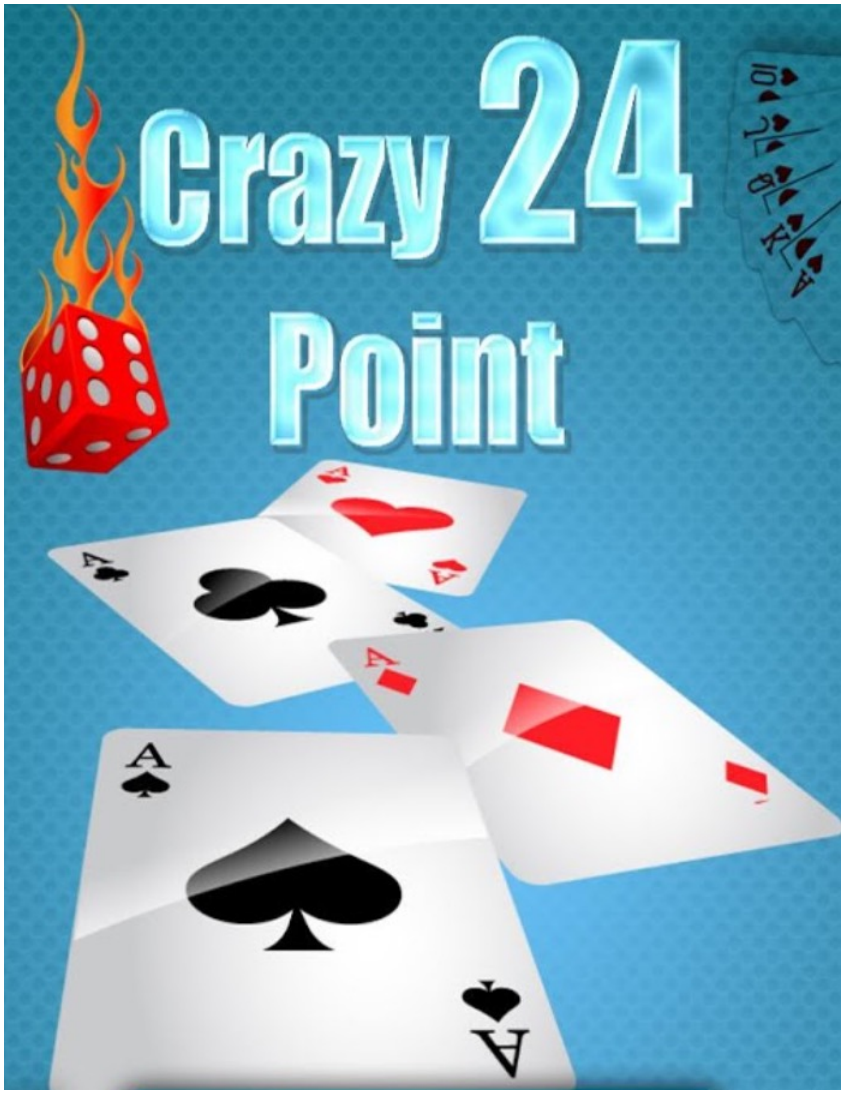
24 Game

- Previous CSCI3100 course project.

Web-based client-server application

+

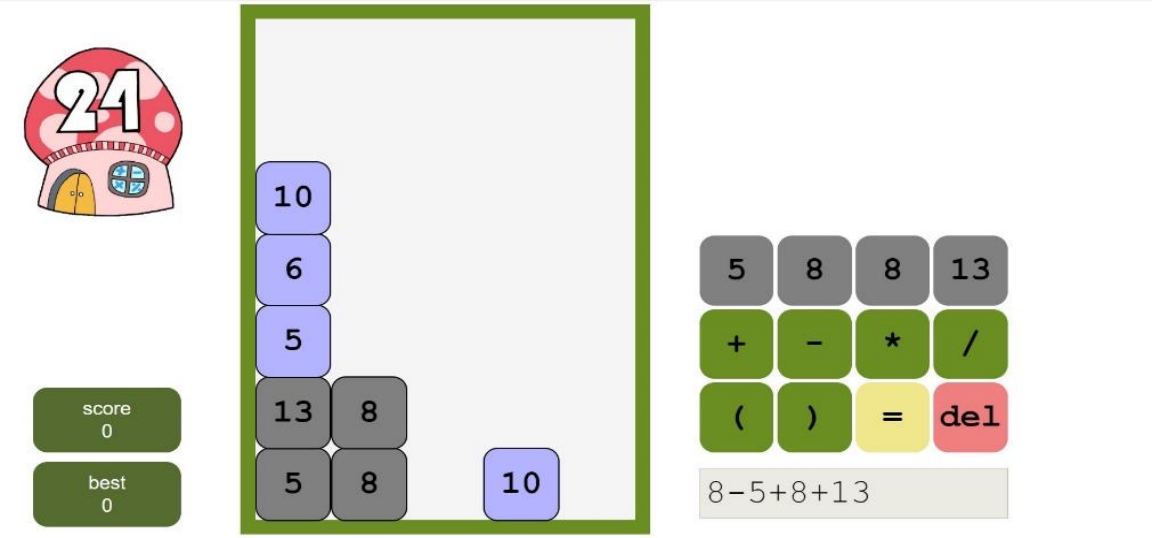
A traditional card game called “Calc 24”



24 Game

- Rules
  - Four numbers.
  - Four basic arithmetic operations $+, -, \times, \div$
  - Get a result of 24.
- Work out a solution ASAP.

- Video Demo



♪ Dj Okawari - Halcyon

24 Game

2018-2019 CSCI3100 Project Group 08

CHAN Ka Yi
CHOW Tsz Ching
FUNG Kwan Hang
LUI Wing Kat
SIN Chun Ming

Procedure

Sign Out

New Game

Leader Board

Profile

Username csci3100

Password •••••••• Edit

Email csci3100@make24.com

Accuracy 0.00

Number of questions solved 0

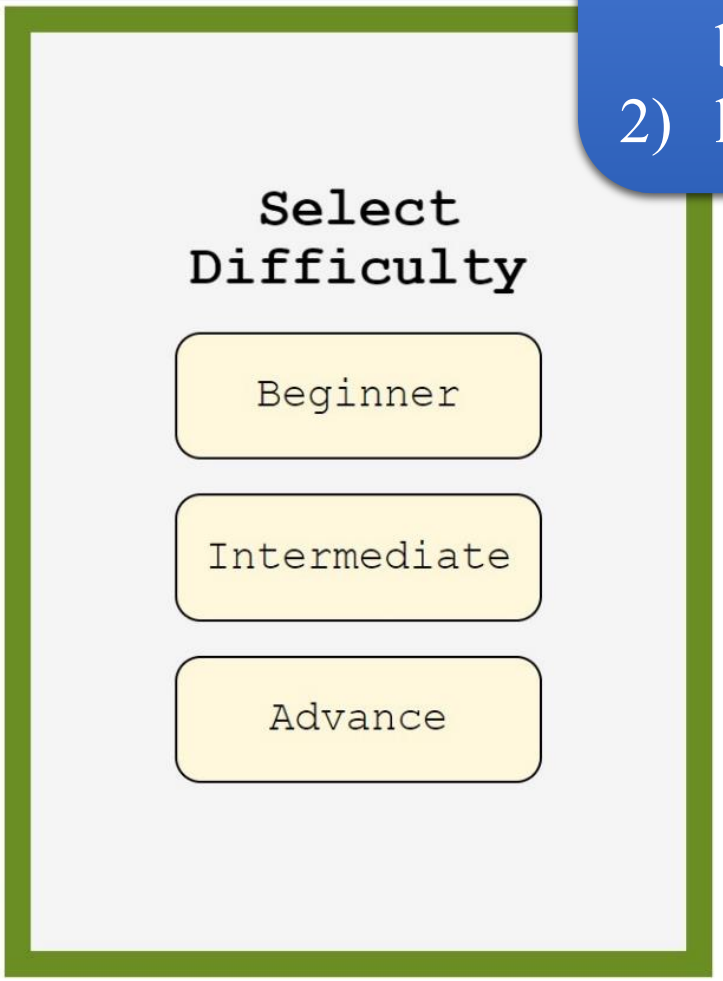
Number of questions encountered 0

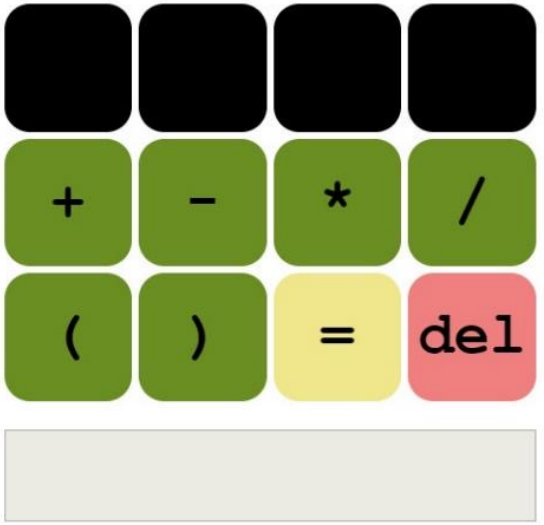
Recent score 0

Best score 0

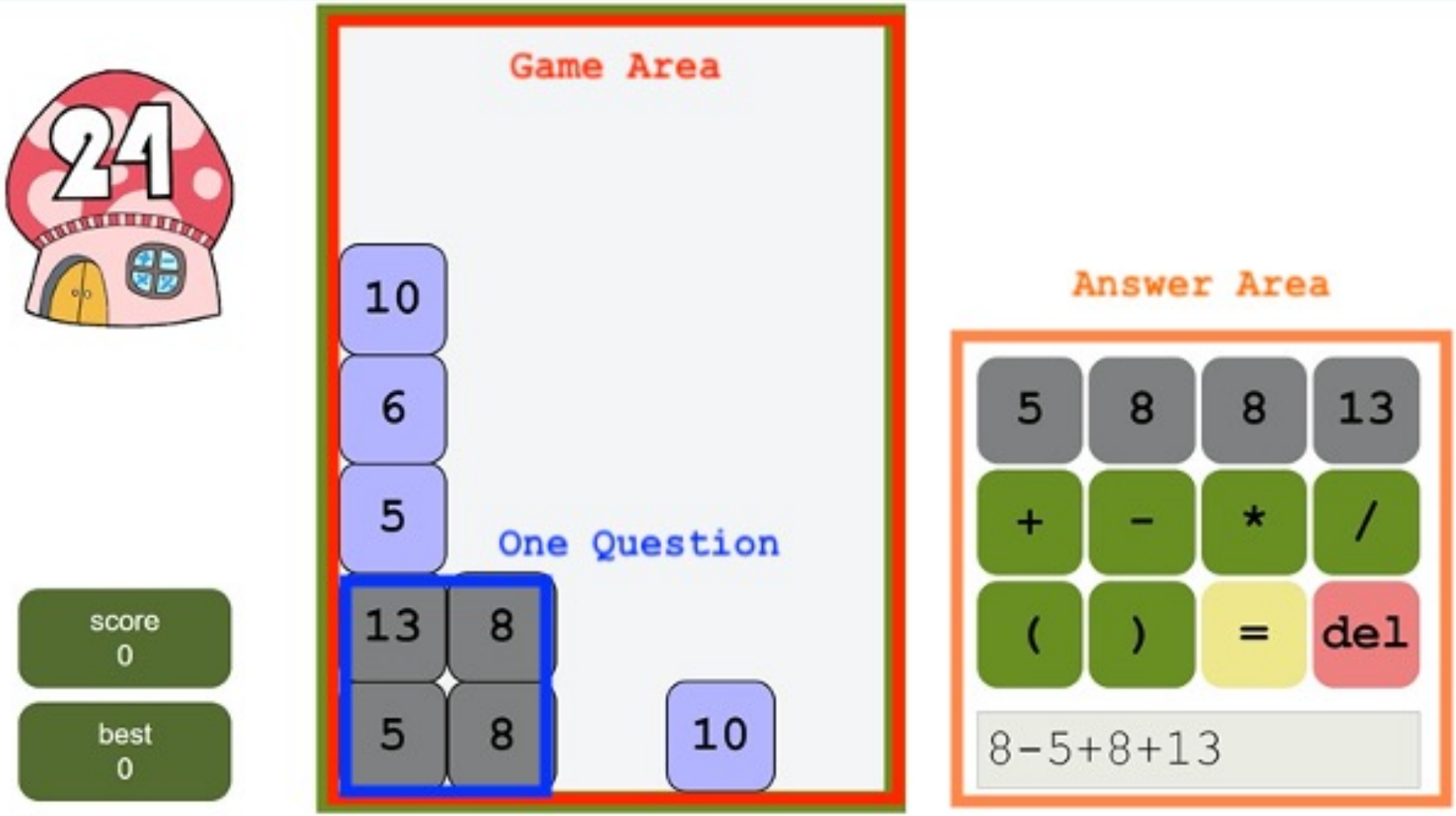
Procedure

The level of difficulty determines
1) how fast (aka. the *velocity*, e.g., one block per second) the blocks fall
2) how often new problems arise.

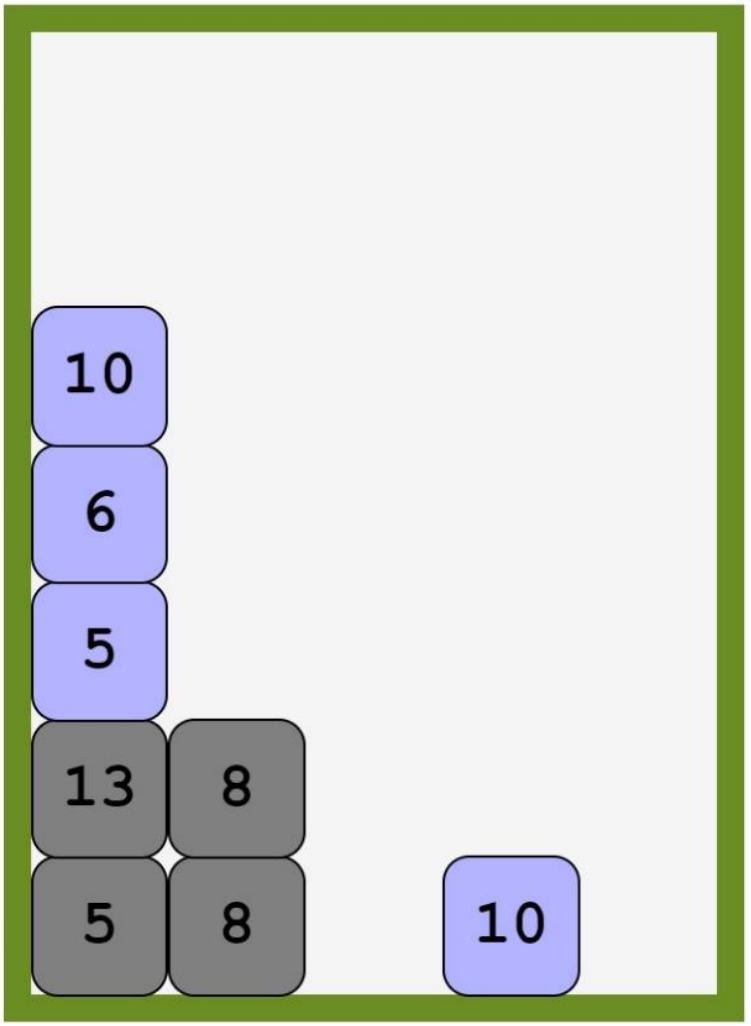


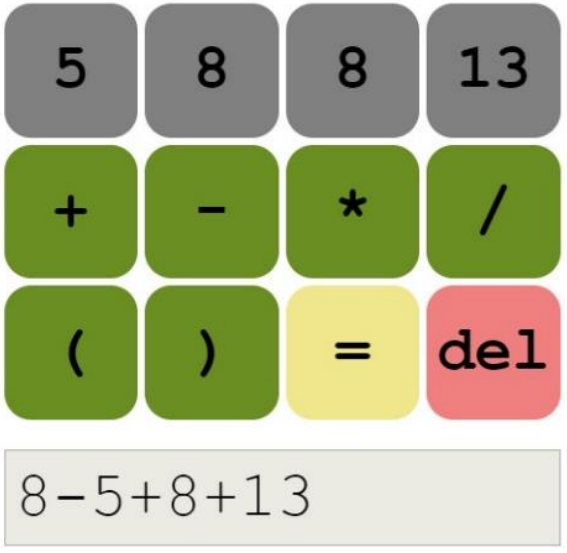


Procedure


24
Game Area
10
6
5
One Question
13
8
5
8
10
Answer Area
5
8
8
13
+
-
*
/
(
)
=
del
8-5+8+13
score
0
best
0

Procedure

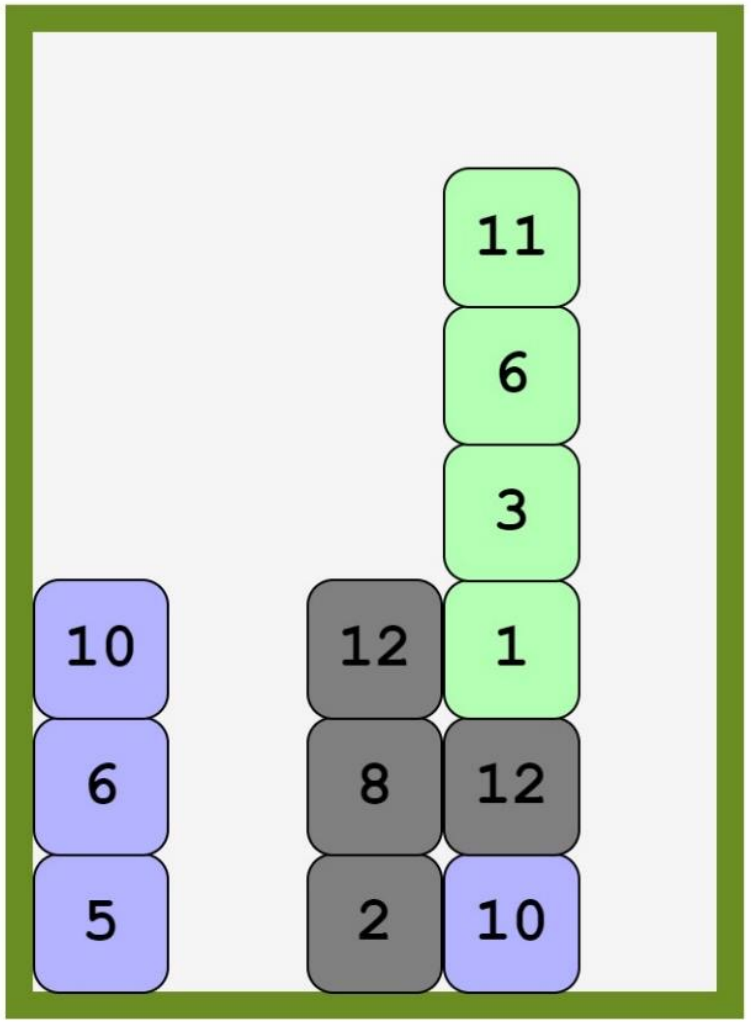


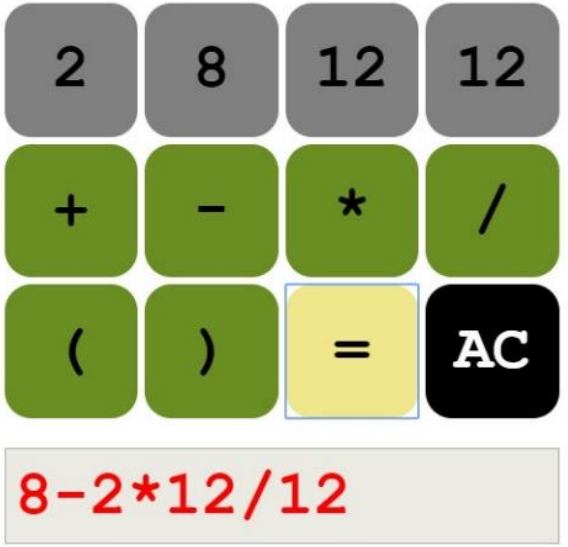


score
0

best
0

Procedure

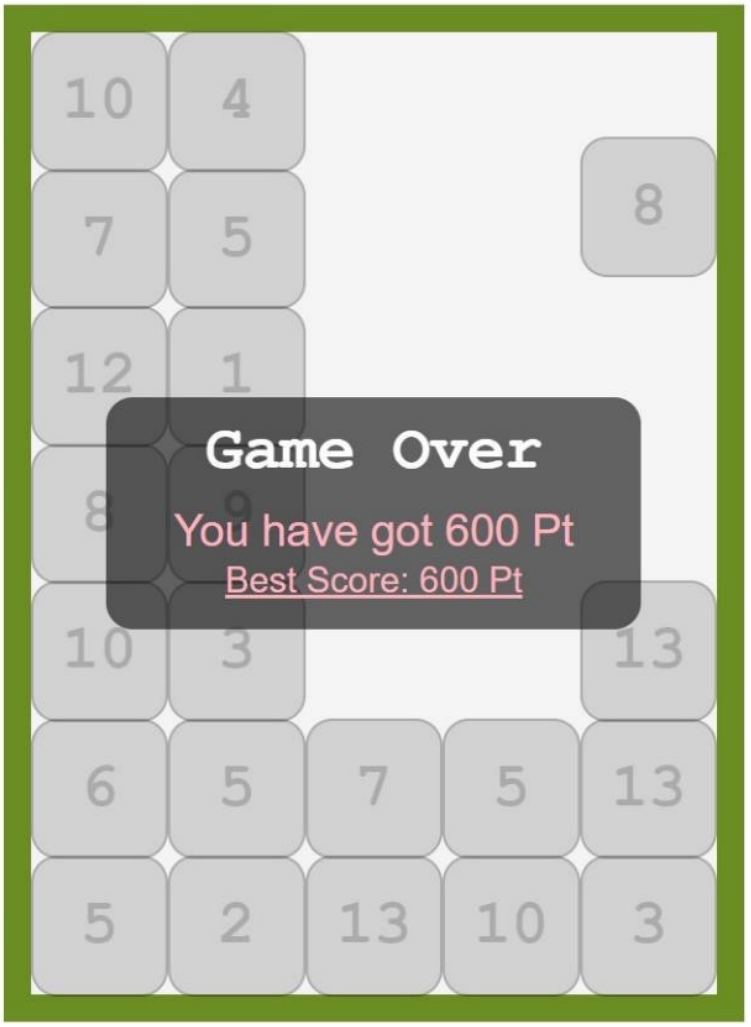


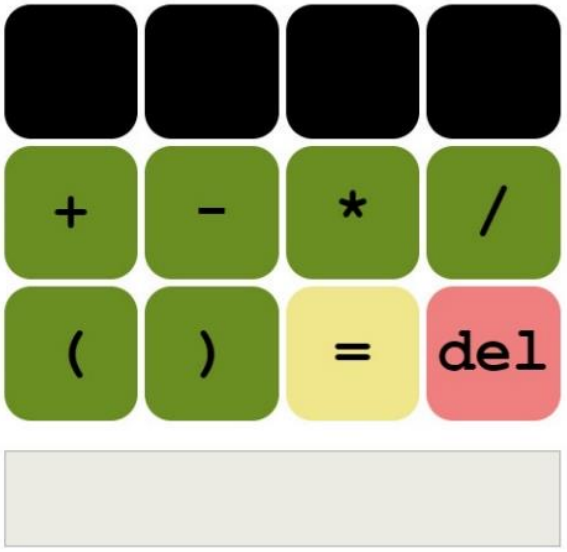


score
100

best
100

Procedure





score
600

best
600

Pseudo code

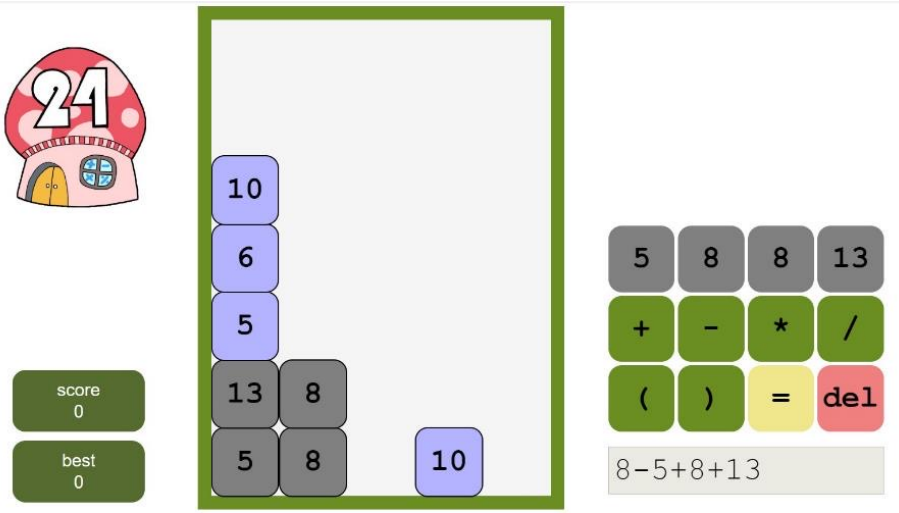
```
connect to user profile database, question
database, and score database
login
display profile
play a new game
   PLAYING
end play game
show score
```

Question (1)

- In the **`PLAYING`** part, what further functions should be considered if we want to design the complete game? Give proper function names and provide brief comments for your functions.

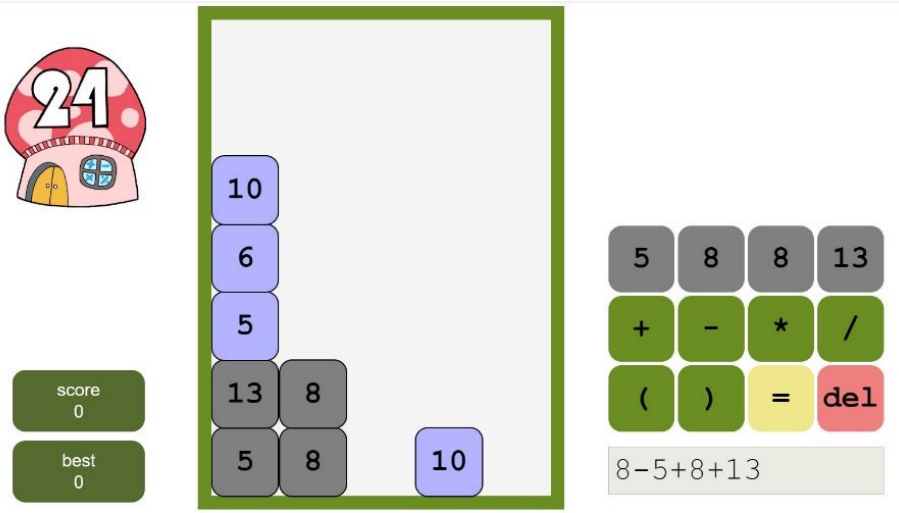
Solution (1)

- chooseDifficulty
  - Choose difficulty level
- getQuestion
  - Get four numbered blocks from the question
- chooseAxis
  - Decide which block fall into which vertical location
- drawBlocks
  - Display blocks of one question
- falling
  - move down dangling blocks every second



Solution (1)

- selectQuestion
  - User select one question to solve by clicking on it
- getAnswer
  - get user's answer from input
- checkAnswer
  - Check whether the answer is right
- eliminateQuestion
  - Remove blocks if the answer is right
- recordScore
  - Record the final score
- gameOver
  - if the blocks reach the top border, the game is over



Question (2)

- Please write the *detailed pseudo code* of **`PLAYING`** with functions you provided in question (1), using *Stepwise Refinement* technique taught in lectures (with at least 3 refinements).

Stepwise Refinement

- a popular method for describing the logical structure of a given algorithm, implemented by a single module

incrementality

Solution (2) : Step 1

```
PLAYING
    chooseDifficulty
    While not gameOver
        PLAY GAME ACCORDING TO SELECTED DIFFICULTY
    End while
    recordScore
END PLAYING
```

Solution (2) : Step 2

PLAYING

chooseDifficulty

While not gameOver

Initiate 2 threads, namely Thread 1 and Thread 2 // ***This is for concurrency, no penalty if you just give the sequential code***

Thread 1 {

getQuestion *// according to selected difficulty*

chooseAxis

drawBlocks

fallDangling *// set falling speed according to selected difficulty*

}

Solution (2) : Step 2

```
        Thread 2 {
                PLAYER ATTEMPT TO SOLVE QUESTION
        }
    End while
    recordScore
END PLAYING
```

Solution (2) : Step 3

Thread 2 {

if selectQuestion

getAnswer

if checkAnswer is right

eliminateQuestion *// After the question is eliminated, blocks of later questions may fall down in Thread 1, thus creating more room in the game area.*

end if

end if

}

Question (3)

- Treating one block as an object, what properties or functions should one block contain? Use *TDN* to design an *abstract data type* module called **`BLOCK_HANDLER`** for block in this game. Please explain your design with proper comments in your TDN.

Abstract Data Type

- Abstract data type is an information hiding module where the representations of data structures are encapsulated.
  - e.g., A “stack” could be defined in only four operations in its interface: push, pop, top, init

TDN: Textual Design Notation

- key sections (clauses)
  - **module** ⇒name
  - **uses** ⇒ relation
  - **exports** ⇒
    - » var, type, procedure (resources)
  - **implementation** ⇒
    - » list internal components

    relation
  - **end**

Solution (3)

Module BLOCK_HANDLER

exports

type BLOCK: ?;

This is an abstract data type module; the data structure is a secret hidden in the implementation part.

function GET_NUMBER(B: in BLOCK): INT;

function GET_VELOCITY(B: in BLOCK): INT;

function GET_X-AXIS(B: in BLOCK): INT;

Solution (3)

function GET_Y-AXIS(B: in BLOCK): INT;

procedure INIT(B: out BLOCK);

Initialize x, y, number, and velocity of the block

procedure FALL(B: in out BLOCK);

FALL represents the procedure of a block falling down

end BLOCK _HANDLER

Question (4)

- Now we have the module for blocks. What properties or functions should a question contain? Use *TDN* to define the *abstract data type* called **`QUESTION_HANDLER`** for question in this game. Please explain your design with proper comments in your TDN.

Solution (4)

Module QUESTION_HANDLER

exports

type QUESTION: ARRAY[1..4] of BLOCK_HANDLER;

This is an abstract data type module; the data structure is a secret hidden in the implementation part.

procedure INIT(Q: out QUESTION);

procedure INIT initializes four BLOCK_HANDLER

Solution (4)

procedure FALL(Q: out QUESTION);

Procedure FALL will call procedure FALL in each BLOCK_HANDLER

procedure DISOLVE(Q: out QUESTION);

Procedure DISOLVE will remove four blocks in the game area

end QUESTION _HANDLER

2. Architecture Design and Design Notation

Source: 2018 / Homework 4 / Question 2

Question

- In this problem, we would like to design a program to detect misspelled words in a text document.
- The spelling checker program should first **split** the document into words, and create a unique word list in order.

Question

- The unique words in this list will be **checked** against the user's dictionary, and unknown words not in the dictionary will be **displayed** for the user to view the misspelled words.
- The user, however, should be allowed to mark some of the unknown words as correctly spelled good words.
- In the end, a new dictionary is created by **merging** these good words in the original dictionary.

Question

a) Using TDN, describe the design of the spelling checker. Concentrate your design in the *module interface*. You should provide comments in the *uses* and *implementation* section to show how they work.

b) Continuing the above TDN, please use GDN to describe the spelling checker.

Solution (1)

module Spellcheck

uses Get_next_word, Add_the_word, Merge

Get_next_word is used to get next word in the document when split document and next word in the unique word list when check against dictionary. Add_the_word is used to add new word to the unique word list and unknown word list and good unknown words. Merge is used to merge the good unknown words with the original dictionary.

exports var B: array (1..2000) of string

type Word: array (1..20) of char

producedure Spellcheck (Dictionary: **in out** B; Document: **in** B)

Spellcheck is a program intended to detect misspelled words in a document.

Solution (1)

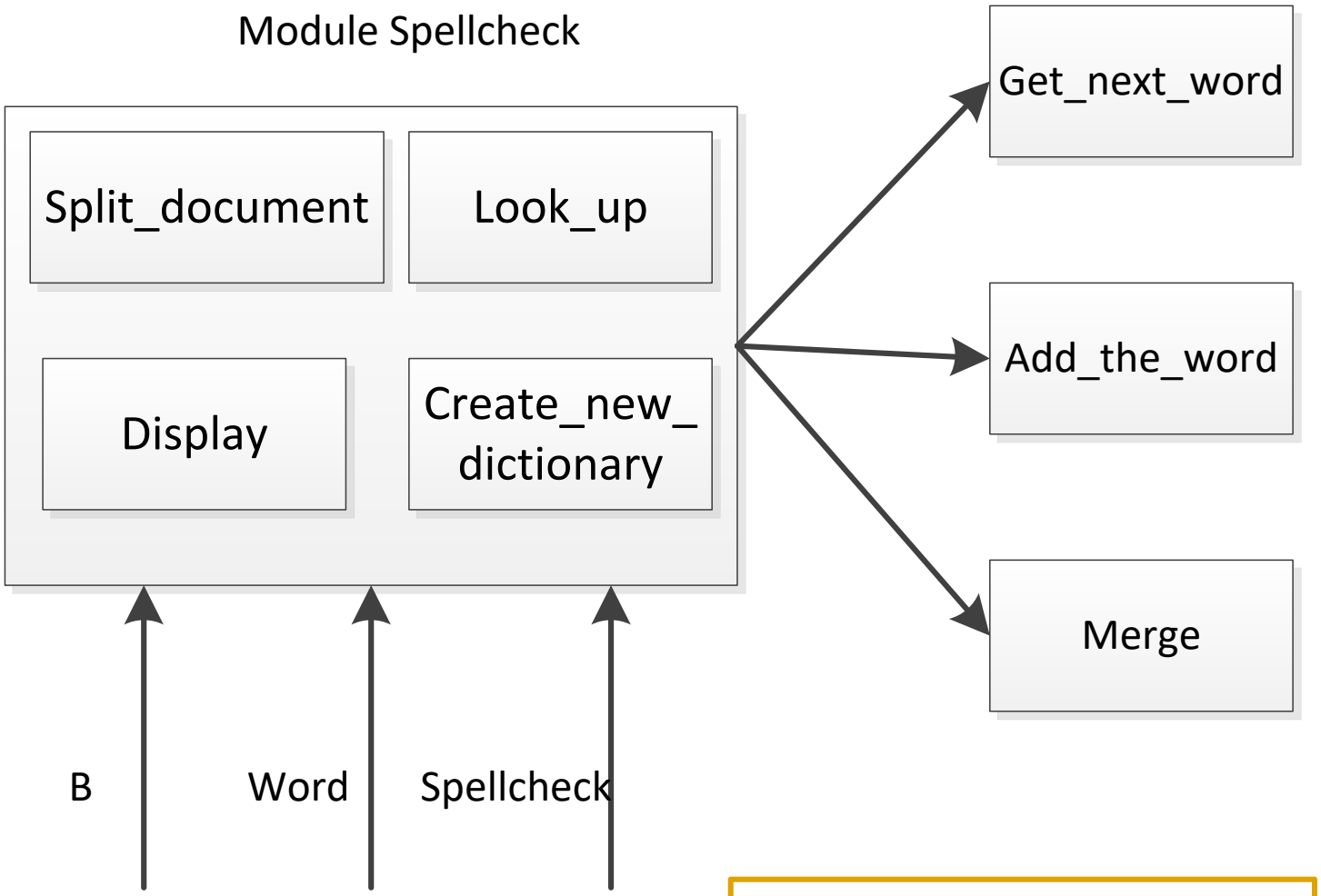
implementation

The program first spit the document, get all unique words in document by eliminating duplicates. Then Look up words in dictionary and find out the unknown words not in the dictionary. Display the unknown words to the user and let him select the good words. Then merge the good words with the dictionary to create a new dictionary.

is composed of Split_document, Look_up, Display, Create_new_dictionary.

end Spellcheck

Solution (2)



Consistency matters!

3. Program Design Technique

Source: 2019 / Homework 4 / Question 3

Jump Game

- Given an array of non-negative integers, you are initially positioned at the first index of the array. Each element in the array represents your maximum jump length at that position. Determine if you are able to reach the last index or not. Note we assume index starts at “0” in these examples.

Example 1

- `Input: [2,3,1,1,4]`
- `Output: true`

Explanation

- One possible way: Jump 1 step from index 0 to 1, then 3 steps to the last index.

Example 2

- `Input: [3,2,1,0,4]`
- `Output: false`

Explanation

- You will always arrive at index 3 no matter what. Its maximum jump length is 0, which makes it impossible to reach the last index.

Question (1)

- Use dynamic programming to solve this problem.

- Hint:
  - As we only jump to the right, if we start from the rightmost side of the array, every time when we query a position to our right, that position has already been determined as being REACHABLE or UNREACHABLE. So we don't need to compute in this case.

Solution (1) : Refinement 1

```cpp
bool canJump(vector<int> &nums) {
    // Initialize state of all index, 0:UNKNOWN, 1:REACHABLE, 2:UNREACHABLE
    // mark the last index as REACHABLE
    for (int i = nums.size() - 2; i >= 0; i--) {
        // Calculate furthest distance of jump
        int furthestJump =
            i + nums[i] < nums.size() - 1 ? i + nums[i] : nums.size() - 1;
        // Iterate through all indexes after i, if there exist an REACHABLE
        // index, mark index i as REACHABLE
    }
    return memo[0] == 1;
}
```

Solution (1) : Refinement 2

```cpp
bool canJump(vector<int> &nums) {
    vector<int> memo(nums.size(), 0); // Initialize state of all index,
                                      // 0:UNKNOWN, 1:REACHABLE, 2:UNREACHABLE
    *(memo.end() - 1) = 1;            // mark the last index as REACHABLE
    for (int i = nums.size() - 2; i >= 0; i--) {
        int furthestJump =
            i + nums[i] < nums.size() - 1 ? i + nums[i] : nums.size() - 1;
        for (int j = i + 1; j <= furthestJump; j++) {
            if (memo[j] == 1) {
                memo[i] = 1;
                break;
            }
        }
    }
    return memo[0] == 1;
}
```

Time Complexity $O(n^2)$

Question (2)

- Now the time complexity of your algorithm is restricted to O(n). Describe your idea first and then implement it in C/C++ or pseudocode.

- Hint: use greedy algorithm to solve this question.

Solution (2)

- Use greedy algorithm to solve this problem.

```cpp
bool canJump(vector<int> &nums) {
    int lastPos = nums.size() - 1;
    for (int i = nums.size() - 1; i >= 0; i--) {
        if (i + nums[i] >= lastPos)
            lastPos = i;
    }
    return (lastPos == 0);
}
```

Solution (2)

- Another solution

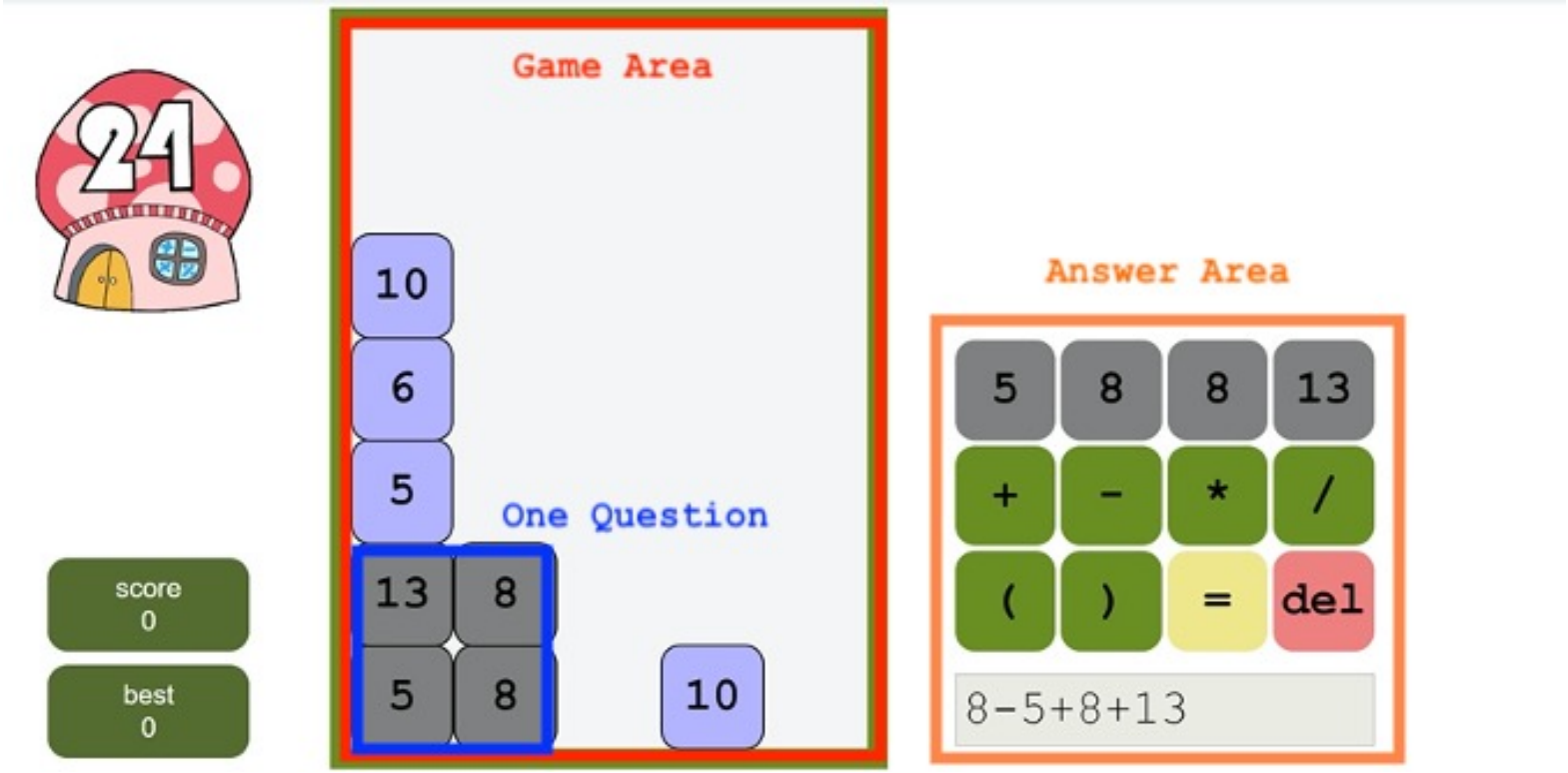
```
1  bool canJump(int *nums, int numsSize) {
2      int distance = 0;
3      for (int i = 0; i < numsSize; ++i) {
4          if (i <= distance) {
5              if (distance < i + nums[i])
6                  distance = i + nums[i];
7
8          } else
9              return false;
10     }
11     return true;
12 }
```

4. Program Design Technique

Source: 2020 / Final / Question 5

Question

- In the Answer Area of 24-Game, user inputs a string and 24-Game will check whether user input equals to 24.



Question

- Let's consider a simplified version of this problem.
- Given a **string** which **only contains numbers ranging from "1" ~ "9" and operators "+-*/"**, you want to write a program to check whether the string expression equals to 24.
- For example
  - TRUE for "3*7+6/2"
  - FALSE for "1+2+3+4"

Question

- **You can assume that the input is valid and does not contain parentheses or 2-digit numbers or “0”.**
- The division operator is the integer division, so you do not need to deal with float number. Please pay attention to the order of operators. “*” and “/” have higher priority than “+” and “-”.

- The following function checks whether the input expression equals to 24, using two stacks. Fill out the 8 numbered blanks in the following function.

```
/**
 * @param a, b operator
 * @return 0 if a has a higher or equal priority than b, e.g.
precede('*', '+') == 0
 * @return 1 if b has a higher priority than a, e.g. precede('+', '*')
== 1
 */
int precede(char a, char b) {
    if (a == '#') {
        return 1;
    } else if ((a == '+' || a == '-') && (b == '*' || b == '/')) {
        return 1;
    } else {
        return 0;
    }
}
```

```
int operate(int left, int right, char op) {
    switch (op) {
    case '+':
        return left + right;
        break;
    case '-':
        return left - right;
        break;
    case '*':
        return left * right;
        break;
    case '/':
        return left / right;
        break;
    }
}
```

```
bool eval(char expression[]) {
    stack<char> operator_stack; // initiate a stack of char
    stack<int> operand_stack;   // initiate a stack of int
    // push the terminator '#'
    operator_stack.push('#');
    int i = 0;
    int left, right;
    char op;
    do {
        if (expression[i] != '+' && expression[i] != '-' &&
            expression[i] != '*' && expression[i] != '/' &&
            expression[i] != '\0') {
            // is an operand, push to operand stack
            operand_stack.push(___ expression[i] ___ - '0');
            i++;
        } else {
            // is an operator, compare priority
            switch (precede(operator_stack.top(), expression[i])) {
            case 1: // top of operator stack has lower priority
```

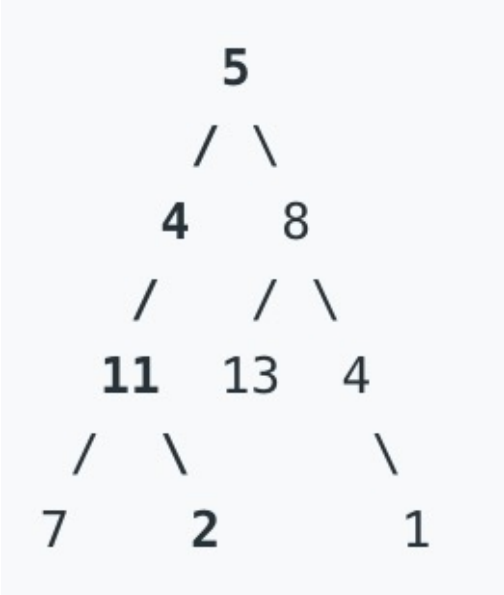
```
                operator_stack.push(___ expression[i] ___);
                i++;
                break;
            case 0: // top of operator stack has higher priority
                ___ right ___ = operand_stack.top();
                operand_stack.pop();
                ___ left ___ = operand_stack.top();
                operand_stack.pop();
                ___ op ___ = operator_stack.top();
                operator_stack.pop();
                ___ operand_stack ___.push(___ operate(left, right, op) ___);
                break;
            }
        }
    } while (i < strlen(expression) ||
             (i == strlen(expression) && operator_stack.top() != '#'));
    return ___ operand_stack.top() ___ == 24;
}
```

5. Software Design: Recursive and Non-Recursive Modules

Source: 2019 / Homework 4 / Question 4

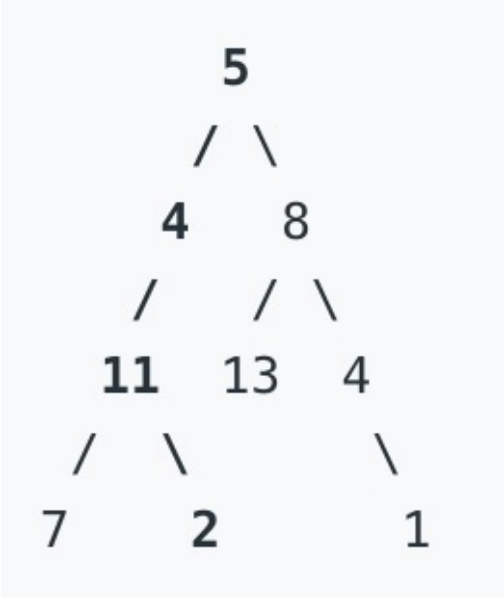
Question

- Given a binary tree and a sum, determine if the tree has a root-to-leaf path such that adding up all the values along the path equals the given sum. All nodes have non-negative values.



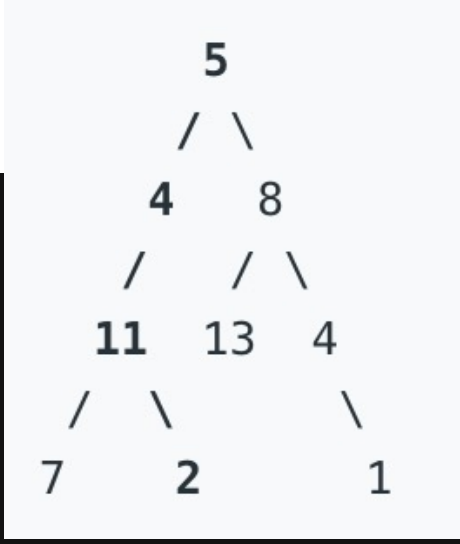
Example

- Given the below binary tree and sum = 22, return true, as there exists a root-to-leaf path 5->4->11->2 which sums to 22.



Question (1)

- The following program is a **recursive** solution. Fill out the numbered blanks in the program and highlight your answer with the corresponding numbers.



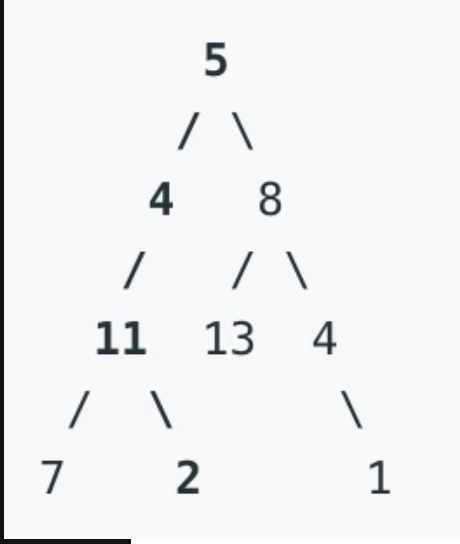

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
bool hasPathSum(TreeNode *root, int sum) {
    if (root == NULL)
        return false; // return false if tree is empty
    sum = sum - root->val;
    if (root->left == NULL && root->right == NULL)
        return (sum == 0);
    return hasPathSum(root->left, sum) || hasPathSum(root->right, sum);
}
```

Question (2)

- Convert the above recursion into a non-recursive version. You may use std::stack to store nodes visited. (Reference: http://www.cplusplus.com/reference/stack/stack/)

```cpp
bool hasPathSum(TreeNode *root, int sum) {
    TreeNode *tmp = root, *last = NULL;
    stack<TreeNode *> s;

    while (tmp || !s.empty()) {
        if (tmp) {
            s.push(tmp);
            sum -= tmp->val;
            tmp = tmp->left;
        } else {
            TreeNode *topNode = s.top();
            if (!(topNode->left) && !(topNode->right) && !sum)
                return true;
            else if (topNode->right && last != topNode->right)
                tmp = topNode->right;
            else {
                last = topNode;
                s.pop();
                sum += topNode->val;
            }
        }
    }
    return false;
}
```



Question (3)

- Recursive solutions are easy to read and write, which is good for software engineering. However, as the tree becomes much larger, the recursive solution generally becomes slower than the non-recursive solution. Acutally the recursive solution may even crash.
- What is the possible reason?
- List at least one method to solve (or optimize) the problem.

- Hint: what happens during a function call?

Hint

- When a function is called, the computer must "remember" the place it was called from, the return address, so that it can return to that location with the result once the call is completed.

- Typically, this information is saved on the ***call stack***.

- Maintaining the call stack cause overhead.

- The space for call stack is limited.

Solution (3)

1. Eliminate useless recursion with carefully designed pruning (early stop).
   - The non-recursive solution do not need to do pruning because variable are stored in another address space, which are generally larger than call stack.

2. Use tailored programming language for recursive problems (e.g. functional programming language). These language manage recursion in a different way.

Thank you!