

BINUS University

Academic Career: <i>Undergraduate / Master / Doctoral *)</i>		Class Program: <i>International/Regular/Smart Program/Global Class*)</i>	
<input checked="" type="checkbox"/> Mid Exam <input type="checkbox"/> Final Exam <input type="checkbox"/> Short Term Exam <input type="checkbox"/> Others Exam : _____		Term : Odd/Even/Short *)	
<input checked="" type="checkbox"/> Kemanggisan <input checked="" type="checkbox"/> Alam Sutera <input type="checkbox"/> Bekasi <input type="checkbox"/> Senayan <input type="checkbox"/> Bandung <input type="checkbox"/> Malang		Academic Year : 2019 / 2020	
Faculty / Dept. : School of Computer Science		Deadline	Day / Date : Wednesday / 15 April 2020 Time : 17:00
Code - Course : COMP6048 – Data Structures		Class :	
Lecturer : Team		Exam Type : Online	
*) <i>Strikethrough the unnecessary items</i>			
<i>The penalty for CHEATING is DROP OUT!!!</i>			

1. There are **2 parts** in this exam, code snippet and case.
2. For code snippet problem:
 - a. you will be given **4 snippets of code** for each given problem e.g. problem1-snippet.cpp etc. There will be some part in the code that **require you to fill it**.
 - b. You are **NOT ALLOWED** either to **change** any pre-defined given code or to **create** your own function.
3. For case problem:
 - a. You will be given an executable file : HovMiningSimulator.exe.
 - b. Solve all the given modules described in the case section below.
4. All the **submission codes are in .cpp files**. Each given problem should have 1 submission code.
5. Please use the given **filename format for your solution**: solution1_nim.cpp, solution2_nim.cpp, solution3_nim.cpp, solution4_nim.cpp, case_nim.cpp.
6. All **your codes should be zipped** using this **format** : nim.zip
7. The exam will be **marked as 0**, if any **plagiarism is found**.

Verified by,

Henry Ham (D5872) and sent to Program on Mar 30, 2020

I. Code Snippet (60%)

1. [15%] Complete the code below to do a hashing using Division & Linear Probing function!

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>

//m = table size
int m = 23;

struct tnode{
    char value[10];
    int step;
}*ND[25];

int hashing(char value[10]){
    //complete the code to do hashing using division hashing
function
    //add all the character ASCII in the string

    return key;
}

struct tnode *newData(char value[10]){
    //complete the code to create a new tnode name ND

    return ND;
}

void insert(char value[10]){
    struct tnode *N_Data = newData(value);
    int key = hashing(value);

    //complete the code to insert new data to hash table. If there
is any collision, use linear probing to solve it and show 'TABLE
IS FULL!!!' when the hash table is full or the data cannot be
insert to the table.
}

void view(){
    for(int i=0;i<m;i++){
        if(ND[i]) printf("[%d]\t%s (%d step(s))\n", i, ND[i]-
>value, ND[i]->step+1);
        else printf("[%d]\tNULL\n", i);
    }
}
```

Verified by,

Harry Ham (D5872) and sent to Program on Mar 30, 2020

```

int main(){
    for(int i=0;i<m;i++){
        ND[i] = NULL;
    }

    insert("AAAAA");
    insert("BBBBB");
    insert("CCCCC");
    insert("AAABB");
    insert("BABAA");
    insert("BAABA");
    insert("BBAAA");
    insert("ABBAA");

    view();
    return 0;
}

```

Output example:

```

[0] NULL
[1] NULL
[2] NULL
[3] AAAAA (1 step(s))
[4] NULL
[5] AAABB (1 step(s))
[6] BABAA (2 step(s))
[7] BAABA (3 step(s))
[8] BBBBB (1 step(s))
[9] BBAAA (5 step(s))
[10] ABBAA (6 step(s))
[11] NULL
[12] NULL
[13] CCCCC (1 step(s))
[14] NULL
[15] NULL
[16] NULL
[17] NULL
[18] NULL
[19] NULL
[20] NULL
[21] NULL
[22] NULL

-----
Process exited after 0.006762 seconds with return value 0
Press any key to continue . . .

```

Verified by,

Henry Ham (D5872) and sent to Program on Mar 30, 2020

2. **[15%]** Thom is a well-known train driver at BINUS's Locomotive Company. His job is to manage goods for every wagon train. Currently, BINUS has two train stations: Anggrek and Syahdan. Moreover, it has four wagon trains: the Polar Express, the Bachmann Penn, the Santa Fe, and the Hogwarts Express.

A wagon train has a connected series of **NINE** cargos that move along the track. Each cargo has a cargo id, which is a combination of a character and two digits number. The cargo of each wagon train is listed below:

Polar Express	= A12 :: B23 :: C35 :: A47 :: B56 :: C66 :: A78 :: B84 :: C92
Bachmann Penn	= A12 :: C35 :: B56 :: A78 :: C92 :: B23 :: A47 :: C66 :: B84
Santa Fe	= C92 :: B84 :: A78 :: C66 :: B56 :: A47 :: C35 :: B23 :: A12
Hogwarts Express	= A12 :: B23 :: C35 :: A47 :: B56 :: C66 :: A78 :: B84 :: C92

Anggrek's and Syahdan's train station can depart a **MAXIMUM** of two trains at once. Furthermore, the train that has the **SAME COMBINATION** of cargo, e.g., Polar Express and Hogwarts Express, **CANNOT** depart together in the same schedule.

You need to help Thom to create a program which able to detect train with identical connected series of cargos. If it is similar, print "1". Otherwise, print "0". Give your answer by completing the given snippet code.

```
#include<stdio.h>
#include<stdlib.h>

struct wagonTrain{
    char cargoType;
    int cargoID;
    struct wagonTrain *next;
};

struct wagonTrain *pushCargo(struct wagonTrain *WT, char cargoType, int cargoID){
    struct wagonTrain *newCargo = (struct wagonTrain*)malloc(sizeof(struct wagonTrain));
    newCargo->cargoType = cargoType;
    newCargo->cargoID = cargoID;
    newCargo->next = NULL;

    if(WT == NULL){
        WT = newCargo;
    }
    else{
        struct wagonTrain *curr = WT;
        while(curr->next != NULL) curr = curr->next;
        curr->next = newCargo;
    }
    return WT;
}
```

Verified by,

Harry Ham (D5872) and sent to Program on Mar 30, 2020

```

void printWagonTrain(struct wagonTrain *WT){
    struct wagonTrain *curr = WT;
    while(curr != NULL){
        if(curr == WT)
            printf("%c%d", curr->cargoType, curr->cargoID);
        else if(curr->next == NULL)
            printf(" :: %c%d\n", curr->cargoType, curr->cargoID);
        else
            printf(" :: %c%d", curr->cargoType, curr->cargoID);
        curr = curr->next;
    }
}

void printAllTrainSet(struct wagonTrain *polarExpress, struct
wagonTrain *bachmannPenn, struct wagonTrain *santaFe, struct wagonTrain
*hogwartsExpress){
    printf("LIST OF TRAIN SET\n");
    printf("=====\n");

    printf("%-17s %s", "Polar Express", "= ");
    printWagonTrain(polarExpress);

    printf("%-17s %s", "Bachmann Penn", "= ");
    printWagonTrain(bachmannPenn);

    printf("%-17s %s", "Santa Fe", "= ");
    printWagonTrain(santaFe);

    printf("%-17s %s", "Hogwarts Express", "= ");
    printWagonTrain(hogwartsExpress);
}

// [#] INSERT YOUR CODE HERE, YOU NEED TO COMPLETE THIS FUNCTION WITH
YOUR ANSWER
int checkWagonTrainCargo(struct wagonTrain *A, struct wagonTrain *B){

    return 0;
}

int main(){
    int i = 0, j = 0;
    char cargoType[3] = {'A', 'B', 'C'};
    int cargoIDType[10] = {12, 23, 35, 47, 56, 66, 78, 84, 92};

    // TRAIN SET 1 - Polar Express
    struct wagonTrain *polarExpress = NULL;
    for(i=0 ; i<9 ; i++)
        polarExpress = pushCargo(polarExpress, cargoType[i%3],
cargoIDType[i]);

    // TRAIN SET 2 = Bachmann Penn
    struct wagonTrain *bachmannPenn = NULL;
    for(i=0 ; i<9 ; i+=2)
        bachmannPenn = pushCargo(bachmannPenn, cargoType[i%3],
cargoIDType[i]);
    for(j=1 ; j<9 ; j+=2)
        bachmannPenn = pushCargo(bachmannPenn, cargoType[j%3],
cargoIDType[j]);
}

```

Verified by,

Henry Ham (D5872) and sent to Program on Mar 30, 2020

```

// TRAIN SET 3 = Santa Fe
struct wagonTrain *santaFe = NULL;
for(i=8 ; i>=0 ; i--)
    santaFe = pushCargo(santaFe, cargoType[i%3],
cargoIDType[i]);

// TRAIN SET 4 = Hogwarts Express
struct wagonTrain *hogwartsExpress = NULL;
hogwartsExpress = pushCargo(hogwartsExpress, 'A', 12);
hogwartsExpress = pushCargo(hogwartsExpress, 'B', 23);
hogwartsExpress = pushCargo(hogwartsExpress, 'C', 35);
for(i=3 ; i<9 ; i++)
    hogwartsExpress = pushCargo(hogwartsExpress,
cargoType[i%3], cargoIDType[i]);

// A FUNCTION TO SEE ALL TRAIN SET
// printAllTrainSet(polarExpress, bachmannPenn, santaFe,
hogwartsExpress);

// POLAR EXPRESS & BACHMANN PENN
printf("%d\n", checkWagonTrainCargo(polarExpress, bachmannPenn));

// SANTA FE & HOGWARTS EXPRESS
printf("%d\n", checkWagonTrainCargo(santaFe, hogwartsExpress));

// HOGWARTS EXPRESS & POLAR EXPRESS
printf("%d\n", checkWagonTrainCargo(polarExpress,
hogwartsExpress));

getch();
return (0);
}

```

Output example:

```

0
0
1

```

Verified by,

Hanry Ham (D5872) and sent to Program on Mar 30, 2020

3. [15%] Hashing Chaining

Hashtable have **31 key**. All the string will use hash function to hash a string into a key. If the key already have data, then new string will be **added to the last linked list** on that key. After all the string already store in hashtable, then **show all the data**. Only show the data if the key has data. Don't forget the **delete all the data** in the end of program. Please refer to chaining code and complete the code. When the program run, it should have the output below:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

struct data
{
    char value[100];
    struct data *next;
};

struct data *hashTables[31];

int hash(const char *str)
{
    int length = strlen(str);
    int base = 11;
    int MOD = 31;
    int key = 0;
    for(int i = 0; i < length; i++)
    {
        key = (key * base) + (str[i] - 'a' + 1);
        key = key % MOD;
    }
    return (key * base) % MOD;
}

void chaining(int idx,const char *str)
{
    struct data *newData = (struct data *) malloc(sizeof(struct
data));
    strcpy(newData->value, str);
    newData->next = NULL;
    //INSERT YOUR CODE HERE, Insert Data into Hashtable with
chaining
}

void viewAll()
{
    //INSERT YOUR CODE HERE, View Data if the index have Data
}

void popAll()
{
    //INSERT YOUR CODE HERE, Pop All the hashtable
}
```

Verified by,

Harry Ham (D5872) and sent to Program on Mar 30, 2020

```

void init()
{
    for(int i = 0 ; i < 31 ; i++)
    {
        hashTables[i] = NULL;
    }
}

int main()
{
    char data[][100] =
    {
        "roti",
        "keju",
        "coklat",
        "durian",
        "nasi",
        "buncis",
        "ayam",
        "air",
        "mie",
        "tahu",
        "tempe",
        "susu",
        "sapi",
        "telur",
        "biskuit",
        "wortel",
        "steak",
        "kentang",
        "apel",
        "melon",
        "ikan"
    };

    init();

    int count = sizeof(data) / sizeof(data[0]);

    for(int i = 0 ; i < count ; i++)
    {
        int idx = hash(data[i]);
        chaining(idx,data[i]);
    }

    viewAll();
    popAll();
    return 0;
}

```

Verified by,

Hanry Ham (D5872) and sent to Program on Mar 30, 2020

Output example:

```
Index 1: apel
Index 2: buncis -> mie
Index 3: durian -> sapi
Index 6: ayam
Index 11: keju -> nasi
Index 13: tahu
Index 14: air
Index 16: roti
Index 18: wortel
Index 22: coklat -> tempe -> melon
Index 23: biskuit
Index 24: susu -> ikan
Index 26: kentang
Index 28: telur
Index 30: steak

-----
Process exited after 0.009004 seconds with return value 0
Press any key to continue . . .
```

Verified by,

Hanry Ham (D5872) and sent to Program on Mar 30, 2020

4. **[15%]** COVID-19 disease has become Pandemic by the World Health Organization and after months of research, the cure namely DIVOCVIRIN has been developed. The next step is mass-producing and distributes to people all over the world. Since the most vulnerable are **the elderly, they will get the cure first.**

You as a programmer asked to develop an application to distribute the cure, here is the details:

1. Application must implement **Priority Queue Double Linked List**
2. Input total patients and total cure available
3. Input patient data (date of birth and full name) as much as total patients
 - a. Format: dd mmmm yyyy - Full Name
 - b. Example: 14 february 1980 - Ronald Rich
4. Push all patient's data to **Priority Queue Double Linked List. The older will be prioritized, check the date of birth.**
5. Pop/delete data as much as total cure
6. View the result:
 - a. If there is enough cure for patients show "All patients get the cure, N cure left"
 - b. **Otherwise**, show "Need N more cure" and show patient's data.
7. Pop all data in **Priority Queue Double Linked List.**

A kind programmer helps you to build the application by giving you snippet code, and you can **continue the code by completing the 3 functions** to run the application. **See the snippet code file for more details.**

Input:

First line consists of Total patients and Total cure

Next Total patients' lines consist of patient's data

Output:

First line is the message "All patients get the cure, N cure left" or "Need N more cure".

Next line consists of patient's data that didn't get the cure.

Verified by,

Hanry Ham (D5872) and sent to Program on Mar 30, 2020

Input Example:

```

7 5
2 august 1970 - Virginia Walter
14 february 1980 - Ronald Rich
18 december 1965 - Camron
30 july 1990 - Rosie Hawkins
1 august 1970 - Yvonne
28 january 1985 - Safa Daly
4 november 1991 - Lorcan Craig Montes

```

Output Example:

```

Need 2 more cure
30 july 1990 - Rosie Hawkins
4 november 1991 - Lorcan Craig Montes

```

```

#include<stdio.h>
#include<stdlib.h>
#include<string.h>

struct data {
    int date, month, year;
    char name[100];
    struct data *next, *prev;
}*head=NULL, *tail=NULL, *temp=NULL;

void popHead();
void popAll();
void printAll();
void addData();
void priorityPush(int, int, int, char*);
int getMonthNumber(char*);
char* getMonthName(int);
long getDateNumber(int, int, int);
struct data *createNode(int, int, int, char*);

long getDateNumber(int date, int month, int year) {
    return (long) (year*10000)+(month*100)+date;
}

int getMonthNumber(char *month) {
    if(!strcmp(month, "january"))return 1;
    if(!strcmp(month, "february"))return 2;
    if(!strcmp(month, "march"))return 3;
    if(!strcmp(month, "april"))return 4;
    if(!strcmp(month, "may"))return 5;
    if(!strcmp(month, "june"))return 6;
    if(!strcmp(month, "july"))return 7;
    if(!strcmp(month, "august"))return 8;
    if(!strcmp(month, "september"))return 9;
    if(!strcmp(month, "october"))return 10;
    if(!strcmp(month, "november"))return 11;
    if(!strcmp(month, "december"))return 12;
}

```

Verified by,

Henry Ham (D5872) and sent to Program on Mar 30, 2020

```

char* getMonthName(int month) {
    switch(month) {
        case 1: return "january";
        case 2: return "february";
        case 3: return "march";
        case 4: return "april";
        case 5: return "may";
        case 6: return "june";
        case 7: return "july";
        case 8: return "august";
        case 9: return "september";
        case 10: return "october";
        case 11: return "november";
        case 12: return "december";
    }
}

void popAll() {
    while(head)
        popHead();
}

void printAll() {
    temp = head;
    while(temp != NULL) {
        printf("%d %s %d - %s\n", temp->date,
getMonthName(temp->month), temp->year, temp->name);
        temp = temp->next;
    }
}

void addData() {
    int date, year;
    char month[100], name[100];
    scanf("%d %s %d - %[^\\n]", &date, &month, &year, &name);
    getchar();
    priorityPush(date, getMonthNumber(month), year, name);
}

struct data *createNode(int date, int month, int year, char *name)
{
    // [1] INSERT YOUR CODE HERE
}

void priorityPush(int date, int month, int year, char *name) {
    // [2] INSERT YOUR CODE HERE
}

void popHead() {
    // [3] INSERT YOUR CODE HERE
}

int main() {
    int totalPatients, totalCure;
    scanf("%d %d", &totalPatients, &totalCure); getchar();
    for(int i=0; i<totalPatients; i++)
        addData();

    for(int i=0; i<totalCure; i++)
        popHead();
}

```

Verified by,

Henry Ham (D5872) and sent to Program on Mar 30, 2020

```
        if(totalPatients < totalCure || totalPatients == totalCure)
            printf("All patients get the cure, %d cure left\n",
totalCure-totalPatients);
        else if(totalPatients > totalCure)
            printf("Need %d more cure\n", totalPatients-
totalCure);

        printAll();
        popAll();
    }
```

Verified by,

Hanry Ham (D5872) and sent to Program on Mar 30, 2020

II. Case (40%)

Hov Mining Simulator

Hov Mining Simulator is a classic console game developed using **C programming language**. In this game the player is role-playing as manager of a **gold mining site**. The mining site can have up to **999 caves**, which **indexed** by a number **between 1 and 999**. The game implements **Binary Search Tree (BST)** data structure to store the cave data using its **index as the key**. Based on the input order of the record, each cave position in the tree representation will be different. As the consequences, each cave will have their own **depth** value. The cave in the **surface (root)**, has **depth value of 1**, while **its child has depth value of 2** and so on. Each cave stores its **total gold production**.

Repetitive insert of the **same cave index** record will **increase its total gold production**. Please check on below illustration. For example, using the given data and inputted into the system by following this order:

- Insert cave 52, gold production 10
- Insert cave 31, gold production 15
- Insert cave 27, gold production 25
- Insert cave 40, gold production 11
- Insert cave 79, gold production 89
- Insert cave 65, gold production 4
- Insert cave 82, gold production 2
- Insert cave 94, gold production 51
- Insert cave 40, gold production 13

Therefore, the BST representation of the data is shown in **Figure 1**. Note that there are duplicate records for cave 40, therefore the later one will update the current gold production.

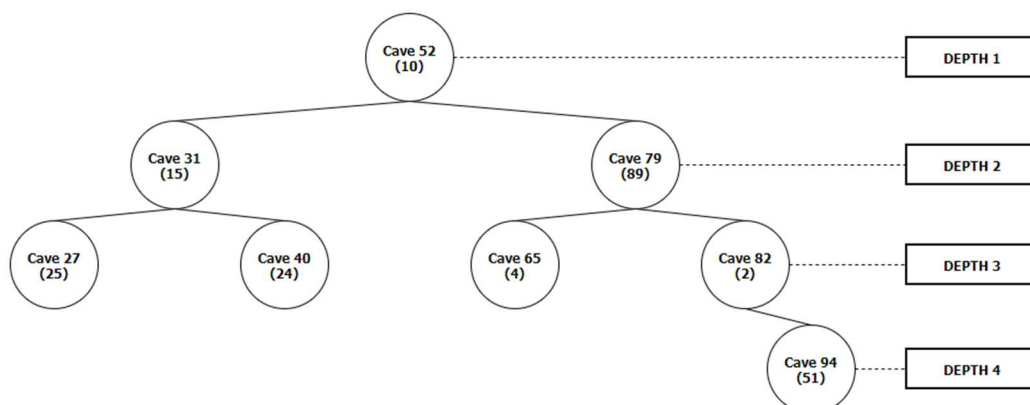


Figure 1. Binary Search Tree Representation of the Record

Verified by,

Hanry Ham (D5872) and sent to Program on Mar 30, 2020

Your task is to re-create this game by following the given requirement. The game has four main menus as shown in **Figure 2**, they are: **[1] Insert Mining Data**, **[2] Display All Cave Data**, **[3] Display Mining Reports**, and **[4] Exit**. Each menu will be explained in each separated chapter. Note that each menu has different weight for the marking purpose.



```

E:\HovMiningSimulator.exe

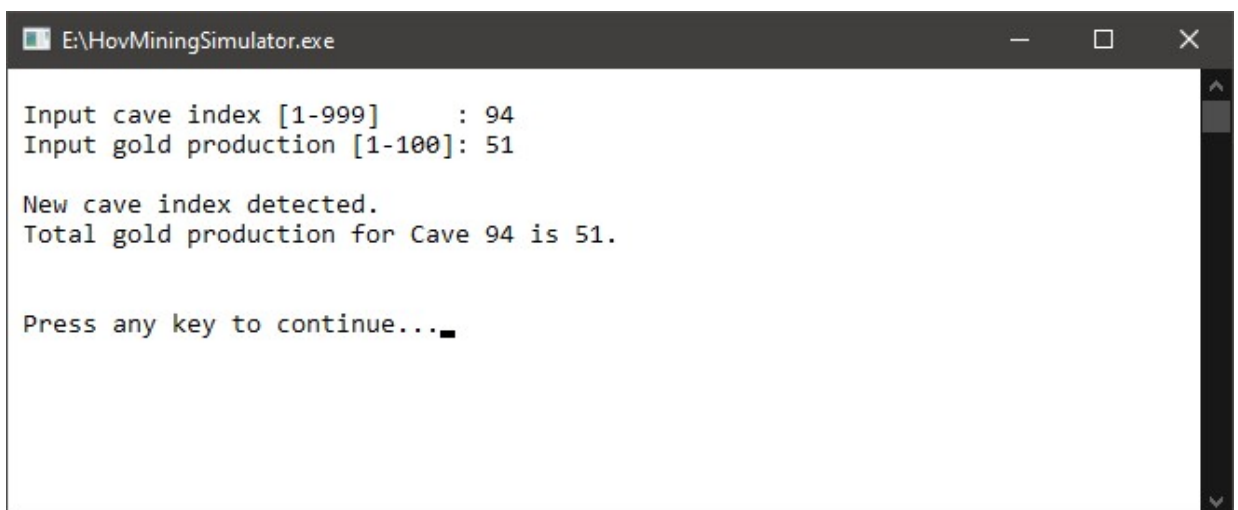
Hov Mining Simulator
=====
[1] Insert Mining Data
[2] Display All Cave Data
[3] Display Mining Reports
[4] Exit
>>

```

Figure 2. Main Menu of Hov Mining Simulator

1. Insert Mining Data (50%)

- ✓ This menu is used to insert mining record into the game memory
- ✓ User will input **cave index** (validate that the value must be **between 1 and 999**) and **gold production** (validate that the value must be **between 1 and 100**)
- ✓ Insert the record into the system and show the status message as shown in **Figure 3** and **Figure 4**.



```

E:\HovMiningSimulator.exe

Input cave index [1-999]      : 94
Input gold production [1-100]: 51

New cave index detected.
Total gold production for Cave 94 is 51.

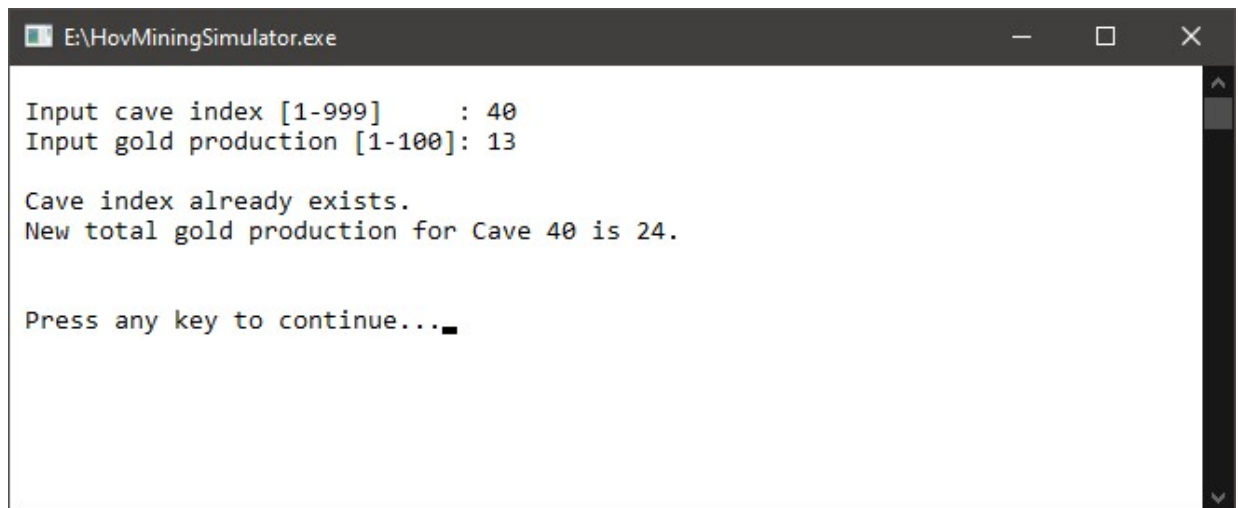
Press any key to continue...■

```

Figure 3. Menu Insert Mining Data when Inserting New Cave Record

Verified by,

Harry Ham (D5872) and sent to Program on Mar 30, 2020



```

E:\HovMiningSimulator.exe

Input cave index [1-999]      : 40
Input gold production [1-100]: 13

Cave index already exists.
New total gold production for Cave 40 is 24.

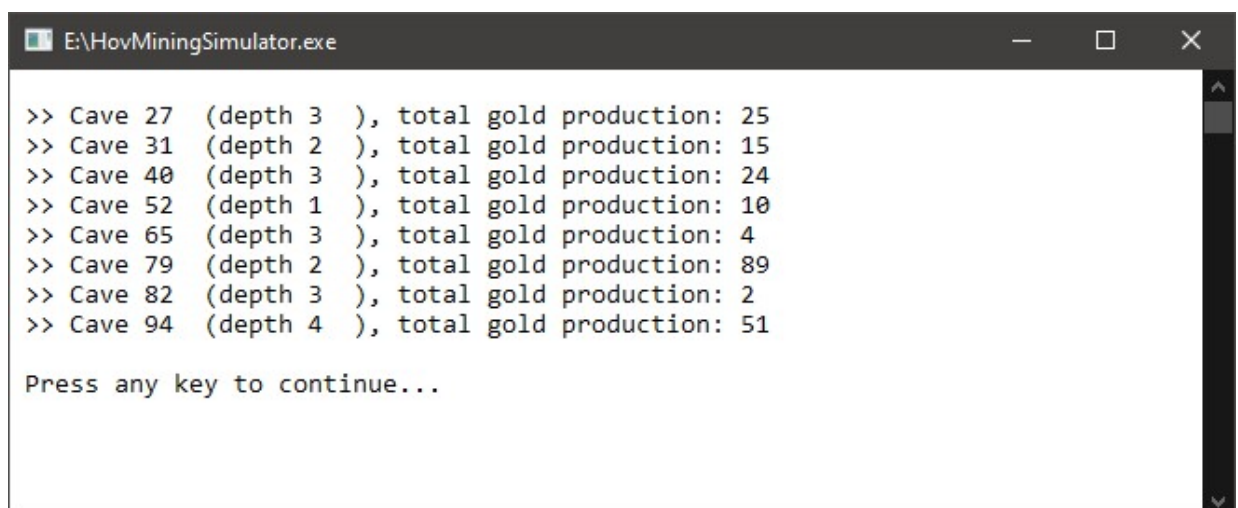
Press any key to continue...

```

Figure 4. Menu Insert Mining Data when Inserting Existing Cave Record

2. Display All Cave Data (20%)

- ✓ This menu is used to display all existing cave record and its total gold production as shown in **Figure 5**. The result is sorted based on cave index in ascending order.
- ✓ Display error message if no record found as shown in **Figure 6**.



```

E:\HovMiningSimulator.exe

>> Cave 27 (depth 3 ), total gold production: 25
>> Cave 31 (depth 2 ), total gold production: 15
>> Cave 40 (depth 3 ), total gold production: 24
>> Cave 52 (depth 1 ), total gold production: 10
>> Cave 65 (depth 3 ), total gold production: 4
>> Cave 79 (depth 2 ), total gold production: 89
>> Cave 82 (depth 3 ), total gold production: 2
>> Cave 94 (depth 4 ), total gold production: 51

Press any key to continue...

```

Figure 5. Menu Display All Cave Data – Non-Empty Record

Verified by,

Henry Ham (D5872) and sent to Program on Mar 30, 2020



Figure 6. Menu Display All Cave Data –Empty Record

3. Display Mining Reports (20%)

- ✓ This menu is used to display total gold production for each level as shown in **Figure 7**. For example, in depth level 3 there are 4 caves: 27 (25 gold), 40 (24 gold), 65 (4 gold), and 82 (2 gold). Therefore, the gold production sum of depth level 3 can be calculated as follow:

$$25 + 24 + 4 + 2 = 55.$$
- ✓ Display the result from the highest to the lowest level of the cave.
- ✓ Display error message if no record found as shown in **Figure 8**.

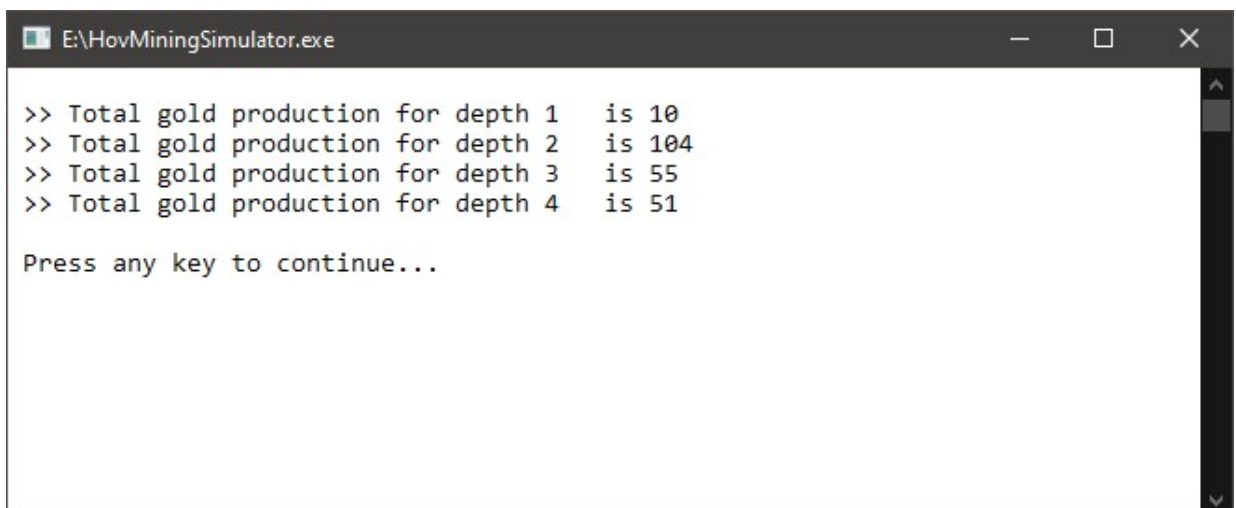


Figure 7. Menu Display Mining Reports - Non-Empty Record

Verified by,

Harry Ham (D5872) and sent to Program on Mar 30, 2020



Figure 8. Menu Display Mining Reports - Empty Record

4. Exit (10%)

- ✓ This menu can be used to exit from the game.
- ✓ Before the program close, it will remove all data in the memory then show a thank you message as shown in **Figure 9**.

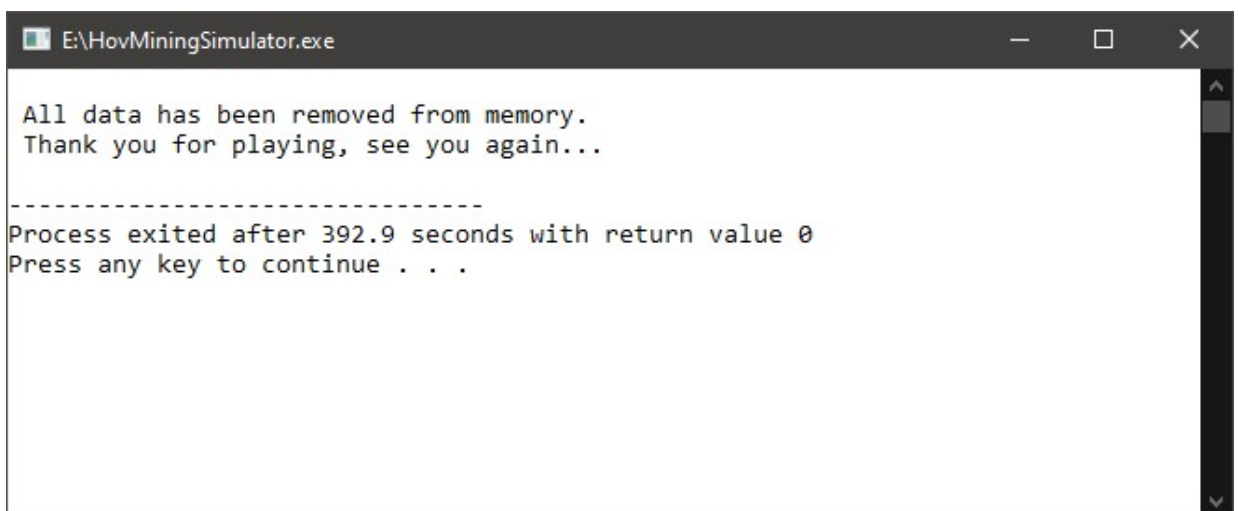


Figure 9. Menu Exit

-- Good Luck --

Verified by,

Hanry Ham (D5872) and sent to Program on Mar 30, 2020