MARIANO MARCOS STATE UNIVERSITY

College of Arts and Sciences

MARIANO MARCOS STATE UNIVERSITY
DEPARTMENT OF COMPUTING AND INFORMATION SCIENCES
BACHELOR OF SCIENCE IN INFORMATION TECHNOLOGY

LODS

(Lab Operationality Discerner System)

SUBMITTED BY:
Asuncion, Carlo Bryant A.

SUBMITTED TO:
MR. WINCHESTER R. GEREZ

SYSTEM THESIS DOCUMENTATION
MAY 2023

INTRODUCTION

In this exponentially evolving digital era students and teachers live through, computer laboratory rooms have become indispensable assets within educational institutions. These spaces serve as hubs for fostering knowledge and granting students access to resources for computing and information. Nonetheless, the task of efficiently managing student attendance and ensuring the optimal functioning of hardware components within these computer lab rooms can present significant challenges. This thesis presents a solution to overcome some of these challenges by developing a C++ system for monitoring attendance and hardware status in the College of Computing and Information Sciences laboratory rooms.

Grappling with recurring incidents of hardware theft within computer lab rooms is quite a nightmare. Equipment such as computer mouse, cables, keyboards, and monitors are particularly vulnerable to theft or malfunction. Furthermore, the conventional method of using handwritten methods to track attendance in these laboratory rooms lacks in efficiency. Consequently, there arises a pressing need for an all-encompassing system capable of seamlessly managing attendance and monitoring hardware in a fully automated fashion.

OBJECTIVES

The main goal of this thesis is to create and implement an automated system that can effectively oversee student attendance and discern the operationality of hardware components in computer laboratory rooms of the college. The proposed system aims to achieve the following objectives:

- The system aims to constantly monitor the status of hardware components after every laboratory session which discerns such information to the facilitators of any unusual activities or discrepancies. By this hardware monitoring system, it will be much more efficient to discern theft and damage to the laboratory equipment.

- The automated attendance system makes the attendance management more efficient and streamlined. Students will be able to check in using their individual identification numbers provided by the university.
- The proposed system will simplify the instructor's task in attendance management and hardware monitoring. It will provide insights in identifying patterns, and help make well-informed decisions based on the status of the used laboratory room.
- The automated system will encourage a sense of accountability among students by the automated attendance records. It will promote punctuality and regularity in lab room usage, benefiting both the students and the department as a whole.

COMPONENTS OF A LAB OPERATIONALITY DISCERNER SYSTEM

1. MAIN MENU

The main menu of LODS (Lab Operationality Discerner System), provides users with options to manage attendance data and monitor the PC hardware status in a given computer lab. It provides a user-friendly interface for navigating through different functionalities of LODS, ensuring convenient access to the desired features. The main menu interface consists of the following:

- **NEW DATA** - Selecting this option allows the user to input new data of student numbers and checks the PC hardware status, to update the system with the latest information.
- **LOAD DATA** - This option enables the user to load previously saved data into the system. It allows for retrieving and accessing existing attendance and hardware records in an accessible text file in the same directory as the program.
- **EXIT PROGRAM -** Choosing this option exits the program, terminating the system and returning '0' the user indicating the operating system that the program terminated successfully.

2. ADD NEW DATA

When adding new data to the program, each individual student using a computer in the laboratory room or the facilitator must enter the student numbers corresponding to the computer being used. The university identification number format can be followed, but to enhance user-friendliness, a dash in the third character element can be omitted or replaced with a space.

Once the student number is entered, it is required to specify the status of the mouse, cables, keyboard, and monitor of the computer each student is currently using. The following character in inputting should be used: 'O' for operational, 'M' for missing, and 'X' if the equipment in a state of malfunction.

In cases where no student is seated at a particular computer, the facilitator of the room can enter '0' to label the area as ''UNUSED_PC''. This helps in accurately tracking the status of each computer and the operationality of the hardware components.

3. LOAD EXISTING DATA

To load existing laboratory room data, the system prompts the user to enter the file name of the desired data file. It is recommended by the program that the file be named in the format of *"[MMDDYY]-[TIME IN]-[TIME OUT].txt",* providing a standardized and organized approach to file naming.

When entering the file name, the user does not need to include the ''.txt'' extension, as the system automatically concatenates it for a smoother and more efficient user experience.

However, it is important to note that if a non-existing file name or a misspelled character is entered by the user, an error prompt will be displayed. This helps ensure that the data loading process, preventing any potential issues caused by incorrect or missing file names.

4. DISPLAY DATA

After the user finishes inputting new data or loading existing data, the program presents a diagram or floor plan view of the laboratory room. In this visual representation, each computer in the room is numbered, and the corresponding student currently using it is displayed alongside the status of the laboratory equipment associated with that student number.

Beneath the diagram, the program offers another selection or menu, guiding the user on the available actions they can perform with the displayed data. The diagram provides a user-friendly and visually appealing interface, making it easier for users to navigate and understand the current occupancy and equipment operationality of the laboratory room.

REFERENCES

Home - Dev-C++ Official Website. (n.d.). https://www.bloodshed.net/

Stroustrup, B. (1994). The Design and Evolution of C++. Addison-Wesley Professional.

Gaddis, T. (2018). Starting Out with C++: From Control Structures through Objects (9th ed.). Pearson.

Prata, S. (2014). C++ Primer Plus (6th ed.). Addison-Wesley Professional.

Patorjk. (n.d.). Text to ASCII Art Generator (TAAG). Retrieved from https://patorjk.com/software/taag

CODE DOCUMENTATION

```
#include<fstream>
#include<iostream>
#include<string>
#include<cstring>
#include<cstdlib>
```

Libraries or header files - The program includes <fstream> for file input and output operations, <iostream> for input and output stream operations, <string> for string manipulation, <cstring> for working with C-style strings, and <cstdlib> for general-purpose operations.

```
using namespace std;
```

Namespace declaration - The "std" namespace contains various standard library components to avoid having to repeatedly type "std::" before each standard library entity.

```
//Lab Operationality Discerner System
struct User{
        string id;
        unsigned char mouse, cable, kybrd, mnitr;
};
```

User struct - represents information about a user in the computer lab, including their "id" and the status of hardware components such as the mouse, cable, keyboard, and monitor. This struct provides a convenient way to store and manage data related to individual users in the lab.

```
enum Menu{
        NEW_DATA=1, LOAD_DATA, EXT_PROGRAM
};
```

Menu enum - defines a set of named constants. In the main menu switch case, it is used to represent the different menu options making the code more readable and helps clarify the purpose of each case.

```
//ID input function and error trapping
void idEnter(char id[], bool& errInput){
bool withDash=false;
	for(int i=0;i<10 && !errInput;i++){
		id[i]=cin.get();
```

idEnter function – input validation function that takes the character array "id" and boolean reference "errInput". It makes use of a for loop and "cin.get()" to input data and stop collecting data when the loop has occurred ten times or if "errInput" becomes false.

```
		//Input stops when user enters newline
		if(id[i]=='\n'){
			id[i]='\0';
			break;
		}
```

When the current character input, "id[i]", is a newline the loop breaks.

```
		if(id[2]==' ' || id[2]=='-') withDash=true;
```

When the third element or "id" index two is either a dash or a space it triggers the boolean variable "withDash" to be true.

```
		//Error trapping if user inputs non-numeric characters
		//That aren't newline nor dash/space at index 2
		else if(cin.fail() || ((id[i]<'0'||id[i]>'9')&&id[i]!='\n')){
			errInput=true;
			cin.clear();
			cin.ignore(1000,'\n');
			cout<<"Please enter a valid student number\n"<<endl;
			for(int j=0;j<10;j++) id[i]='\0';
		}
	}
```

When current character input makes cin.fail() true, or non-numeric characters except newline and space/dash on character element three is entered, the code traps the error with a prompt clearing and ignoring said error, and breaks out of the loop as "errInput" is now true.

```
	//When '0' alone is entered, it will labeled as "UNUSED_PC"
	if(id[0]=='0'&&id[1]=='\0'){
		char none[]="UNUSED_PC";
		strcpy(id, none);
```

When user enters '0' alone, the character array "none" is copied to "id".

```
        } else if(((strlen(id)>8||strlen(id)<8)&&!withDash&&!errInput)||
        ((strlen(id)>9||strlen(id)<9)&&withDash&&!errInput)){
                        errInput=true;
                        if(strlen(id)>8)cin.ignore(1000,'\n');
                        cout<<"Please enter a valid student number\n"<<endl;
        } //Error trapping when input exceeds or is less than the character limit
}
```

When "errInput" is false, and the user enters something larger or shorter than the character limit of a standard university student number, depending on the presence of a dash, the code traps the error.

```
void idFormat(char id[]){
        //A space in index 2/third element will be overwritten with a dash
        if(id[2]==' ') id[2]='-';
        //If user does not input space/dash, this for loop will do it
        else if(id[2]!='-' && id[2]!='U'){
                for(int j=10;j>2;j--){
                        *(id+j)=*(id+j-1);
                } id[2]='-';
        }
}
```

idFormat function – input validation function that takes the character array "id" and makes the user input follow the university student number format, wherein a dash exists in the third character element or the index two of the array.

If the input doesn't contain a space in index two, it takes the pointer of the last element in the character array and dereferences it and move the character before it to the current pointer address. This is done until the third index of the array is move, where the second index is the overwritten by a dash.

```
//Hardware data input function and error trapping
void hardwareErr(char u[], bool& errChar){
```

hardwareErr function – error trapping function that takes a character array and boolean reference "errChar". This function is used inside the function "hardwareEnter" to trap input errors.

```
        //If user enters more than one character
        if(strlen(u)>1){
                errChar=true;
                cout<<"Please input a valid character\t\t\t\t  ";
```

The input in this field must only be a single character. But to trap errors in the situation the user enters more than one, the current input field is a character array. If the user enters a string length of more than one, "errChar" then is turned to true.

```
        } else {
                //Takes the value of the first character in the array
                switch(*u){
                        case 'O':
                        case 'o':
                                *u='O';
                                break;
                        case 'X':
                        case 'x':
                                *u='X';
                                break;
                        case 'M':
                        case 'm':
                                *u='M';
                                break;
                        default:
                                errChar=true;
                                cout<<"Please input a valid character\t\t\t\t  ";
                                //Error trapping when character is none of the above
                }
        }
}
```

When the user enters and character with string length equal to one, the first character of the pointer address in the array is dereferenced and will go on to what case it will be classified as. If the character is not within the specified cases, it will be classified under "default" and be prompted as an error instead.

```
void hardwareEnter(User u[], int i){
```

hardwareEnter function – input validation function that takes a character array and the value of an integer, which in this case, is the current integer value of the for loop inside the main function. This function is used entering the status of the hardware components.

```
bool errChar=false;
for(int j=0;j<4;j++){
        do{     //A character array is dynamically allocated to trap
                //Errors when user enters more than one character
                char *charOpt=new char[10];
                errChar=false;
                if(j==0) cout<<"PC ["<<i+1<<"] Mouse status    : ";
                if(j==1) cout<<"PC ["<<i+1<<"] Cable status    : ";
                if(j==2) cout<<"PC ["<<i+1<<"] Keyboard status : ";
                if(j==3) cout<<"PC ["<<i+1<<"] Monitor status  : ";
                cin.getline(charOpt, 10);
```

A for loop is set to go through the four different hardware equipment specified in the system. A new character array is dynamically allocated to manually manage memory and temporarily store the users' input.

```
                //Error trapping when user enters beyond the limit
                if(cin.fail()){
                        cin.clear();
                        cin.ignore(1000, '\n');
                        errChar=true;
                        cout<<"Please input a valid character\t\t\t\t  ";
                } else hardwareErr(charOpt, errChar);
```

If the user inputs something that triggers cin.fail() to be true, the error will be trapped, otherwise the input will go to the function "hardwareErr" to check for any additional errors.

```
                if((*charOpt=='O' || *charOpt=='X' || *charOpt=='M')&&j<3)
                        cout<<"\t\t\t\t\t\t\t  ";
                if(j==0) u[i].mouse=*charOpt;
                if(j==1) u[i].cable=*charOpt;
                if(j==2) u[i].kybrd=*charOpt;
                if(j==3) u[i].mnitr=*charOpt;
                delete charOpt;
        } while(errChar);
}
```

The first character of array "charOpt" is then passed to the structure members. As long as "errChar" is true the loop will continue to ask the user to input.

```
//Other input functions
void selection1to3(int* option){
```

selection1to3 function – input validation function that takes the pointer address of integer option. There are multiple instances along the program where the user will choose from integers 1,2, and 3; this function takes care of it and prevents the redundancy of writing the same error trapping code.

```
	bool errOption=false;
	do{
		if(errOption){
			cin.clear();
			cin.ignore(1000, '\n');
			cout<<"  Invalid input :(\t";
		}
		errOption=false;
		cout<<"Indicate your option: ";
		cin>>*option;
		if(cin.fail() ||
		(cin.peek()<'1'||cin.peek()>'3')&&(cin.peek()!='\n') ||
		(*option!=1&&*option!=2&&*option!=3))
		//Error trapping when input is neither 1, 2, or 3
		errOption=true;
	} while(errOption);
}
```

The error trapping happens when boolean variable "errOption" is set to true, where the do-while loop continues to loop and clears, ignore and prompts the error message.

The "errOption" only becomes true when the user inputs something that make cin.fail() true, inputs a character that is lesser than 1 and greater than 3 (that is not a newline character), or when the input is not exact in terms of the number of digits.

```
void filenameInput(char filename[]){
```

filenameInput function – input validation function that takes the character array of "filename". The function is used when the user saves newly entered data or when the user loads existing data, wherein it aims to find errors when present.

```
	do{ //Error trapping for file name input
		if(strlen(filename)>24){
			cin.clear();
			cin.ignore(10000, '\n');
			cout<<"\n\tThe character limit is only 24. Please try again.";
		} else if(*filename=='\n') cin.ignore(1); //Prevents input buffering
		cout<<"\n\tEnter the file name: ";
		for(int i=0;i<25;i++){
			filename[i]=cin.get();
			//Stops input when newline is entered by the user
			if(filename[i]=='\n'){
				filename[i]='\0';
			break;
				}
			}
	} while(strlen(filename)>24);
	//File extension is concatenated by the program
	char ext[]=".txt";
	strcat(filename, ext);
}
```

The character limit of this input validation is set to 24, and if the user violates this rule by going over the limit the error will be trapped. The user however does not need to reach the 24 character limit as when input with "cin.get()" is finished and the user presses enter, the for loop will break without any error trapping.

The user does not need to type in the ".txt" part of the file name, as this will be done by the program itself by concatenating character array "ext" to the "filename" character array.

```
void outputMenu(bool& backToMain, int pcQtty, User u[]){
	cout<<"\n\t\t\t\t\tWhat's next?"<<endl;
	cout<<"\t\t\t\t[1] SAVE DATA"<<endl;
	cout<<"\t\t\t\t[2] BACK TO MAIN MENU"<<endl;
	cout<<"\t\t\t\t[3] EXIT PROGRAM"<<endl;
	cout<<"\n\t\t\t";
	int* option=new int;
	selection1to3(option);
```

outputMenu function – a menu function that takes the boolean reference "backToMain", integer value of "pcQtty" and the User struct with variable "u". It appears at the end of inputting new data, letting the user choose options to do after.

```
switch(*option){
        case 1:{
        ofstream fout;
        cout<<"\n\t(Recommended file format ";
        cout<<"\"MMDDYY-[TIME IN]-[TIMEOUT]\", e.g.";
        cout<<" 123122-11-2)";
        char *filename=new char[25]; if(cin.peek()=='\n') cin.ignore(1,'\n');
        filenameInput(filename);
        fout.open(filename);
        if(fout.is_open()){
                cout<<"\n\tCreating "<<filename<<" file..."<<endl;
                for(int i=0;i<pcQtty;i++){
                        fout<<u[i].id<<" "<<u[i].mouse<<" "
                        <<u[i].cable<<" "<<u[i].kybrd<<" "<<u[i].mnitr<<endl;
                }
                fout.close();
                cout<<"\tData has been saved! Please restart the program ";
                cout<<"and load the data to check."<<endl;
        } else cout<<"\tSaving failed! Sorry :("<<endl;
        delete[] filename;
        break;
        }
```

The first option allows the user to save the data with variable "fout", it writes data to the file, indicating success or failure of the operation. It dynamically allocates memory for a character array to store the "filename" and prompts the user for input.

The file is opened for writing, and if successful, data from the array is written to the file. Afterward, the file is closed, and appropriate messages are displayed. Finally, the allocated memory is freed.

```
        case 2: backToMain=true;
        break;
```

The second option makes boolean reference "backToMain" true, which triggers the loop in the main function to go back to function "mainMenu".

```
        case 3:
        break;
}       delete option;
}
```

The third option directly exits the switch; "option" is deallocated afterwards.

```
void inputMenu(bool& backToMain, bool& backToLoad){
	cout<<"\n\t\t\t\t\tWhat's next?"<<endl;
	cout<<"\t\t\t\t[1] LOAD OTHER DATA"<<endl;
	cout<<"\t\t\t\t[2] BACK TO MAIN MENU"<<endl;
	cout<<"\t\t\t\t[3] EXIT PROGRAM"<<endl;
	cout<<"\n\t\t\t";
	int* option=new int;
	selection1to3(option);
```

inputMenu function – a menu function that takes the boolean reference "backToMain" and "backToLoad". It is presented at the end of loading an existing data, letting the user choose options on what to do next.

```
	switch(*option){
		case 1:{
		backToLoad=true;
		system("cls");
		break;
		}
```

The dynamically allocated integer "option" is dereferenced and checks what case it falls under. In case 1, it changes the boolean variable "backToLoad" to true and clears the screen making the main function go back to inputting the file name of existing lab room data.

```
		case 2: backToMain=true;
		break;
```

In case 2, it changes the boolean variable "backToMain" to true which triggers the loop in the main function to go back to function "mainMenu".

```
		case 3:

		break;
	} delete option;
}
```

In case 3, the menu exits the program. After a case is finished the "option" is then deallocated.

```
//Text Art function declaration
void mainMenu();
void diagram(User u[]);
void diagramNull();
```

The functions; ‘‘mainMenu’’, ‘‘diagram’’ which takes User struct in the variable ‘‘u’’, and ‘‘diagramNull’’, are declared here. They serves as functions to visually display text art within the system.

```
int main(){
        //PC Quantity
        unsigned int pcQtty=24;
        User user[pcQtty];
        //Error trapping bool variable
        bool errInput=false;
        //Main menu bool variable
        bool backToMain=false;
```

The main function - the starting point of the system where the execution begins, coordinating the program's objectives, including input/output operations, function calls, and program logic. It serves as the primary control flow for the program and determines how the program behaves and terminates.

Variables such as the PC quantity unsigned integer, ‘‘user’’ variable of User struct, boolean ‘‘errInput’’ and ‘‘backToMain’’, are declared and initialized at the top of the main function.

```
        do{
                backToMain=false;
                system("cls");
                //Program start
                int* option=new int;
                        mainMenu();
                        selection1to3(option);
                //The value of option is converted to it's respective enum member
                Menu selected=static_cast<Menu>(*option);
                delete option;
                switch(selected){
```

An integer pointer ‘‘option’’ is dynamically allocated, which is then converted to its corresponding enum member using ‘‘static_cast’’. Finally, the dynamically allocated memory is released.

```
case NEW_DATA:{
        cout<<"\t\t\tENTER NEW LAB ROOM DATA"<<endl;
        diagramNull();
        char *id=new char[10];
        if(cin.peek()=='\n') cin.ignore(1);
        //Prevents newline input buffering
        for(int i=0;i<pcQtty;i++){
            do{
            errInput=false;
            cout<<"Enter Student Number of user in PC ["<<i+1<<"]: ";
            idEnter(id, errInput);
            } while(errInput);
            idFormat(id);
            user[i].id=id;
            cout<<"\t\t\t\t\t"<<user[i].id<<"\t  ";
            hardwareEnter(user, i);
        } delete id;
        //Visual diagram of data
        system("cls");
        diagram(user);
        outputMenu(backToMain, pcQtty, user);
    break;
    }
```

If the value of "selected" corresponds to "NEW_DATA", the user will be prompted to enter new data with a visual representation of the laboratory room. A character array named "id" is dynamically allocated for it to temporarily store the string input of the user.

A "cin.ignore" is put in place to prevent the event that newline character causes input buffering, after that the for loop will keep track of the number computers needed for the system. "idEnter" function will be called to input the data on the "id" array and keep track of errors. When there's an error, bool "errInput" will be true, and the do-while loop will run again until user inputs something valid.

The student number format is then checked with function "idFormat", afterwards the data from "id" is finally transferred to the user struct "id" string. The user will enter the status of the hard equipment with the use of function "hardwareEnter", completing this will deallocate the "id" array. Finally, the data is visualized with "diagram", and menu function "outputMenu" will appear.

```
case LOAD_DATA:{
        bool backToLoad=false;
        do{
                backToLoad=false;
                cout<<"\t\t\tLOAD COMPUTER LAB ROOM DATA"<<endl;
                ifstream fin;
                char *filename=new char[25];
                for(int k=0;!fin.is_open();k++){
                if(k==0 && cin.peek()=='\n') cin.ignore(1,'\n');
                //Prevents newline input buffering of the first loop
                cout<<"\n\t(Make sure the file is in the same directory as";
                cout<<" the program)";
                filenameInput(filename);
                fin.open(filename);
                if(fin.is_open()){
                        cout<<"Loading "<<filename<<"...";
                        for(int i=0;i<pcQtty;i++){
                        fin>>user[i].id>>user[i].mouse>>user[i].cable;
                                fin>>user[i].kybrd>>user[i].mnitr;
                        }
                } else cout<<"\tUnable to open "<<filename<<", please check";
                                cout<<" properly."<<endl;
                } delete[] filename;
                fin.close();
                //Visual diagram of data
                system("cls");
                diagram(user);
                inputMenu(backToMain, backToLoad);
        } while(backToLoad);
break;
}
```

When the value of "selected" corresponds to "LOAD_DATA", the user will be prompted to enter existing data in the same directory as the program itself.

The user enters the "filename" in the dynamically allocated character array, and attempts to open the file. If the file is successfully opened, the data is read into the for loop, in which it looks for data to fill up the struct members. A visual diagram of the lab room is then displayed, and the user is presented with the "inputMenu". The loop continues unless the user chooses not to wherein "backToLoad" is false.

```
                case EXT_PROGRAM:
                        cout<<"\t\t\tGOODBYE LODS :D"<<endl;
                break;
                }
        } while(backToMain);
        return 0;
}
```

If the value of "selected" corresponds to "EXIT_PROGRAM", a goodbye message will display and the program will end, unless the user chooses to go back to the main menu with one of the boolean reference within the functions which overwrites "backToMain" to true.

```
//Text Art
void mainMenu(){ … }
void diagramNull(){ … }
void diagram(User u[]){ … }
```

These are the function definitions of "mainMenu", "diagram", and "diagramNull"; which consists of more than seventy lines of text-based art code.