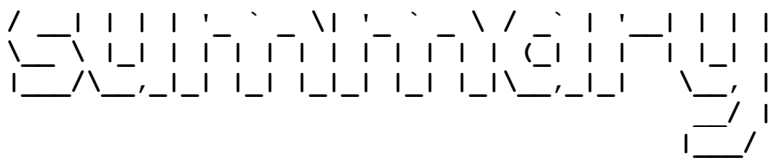
by **Kenzie Vasquez**

CIC-17A-42636

Dr. Mark Lehr

4/17/17



Hangman is a game where the player tries to guess what a mysterious word is by suggesting letters within a number of tries.

The word is first displayed as a row of underscores:

word = _____

and when the player correctly guesses a letter, the letter uncovers itself in its appropriate spot:

player guesses ‘w’

word = w___

Each time the player guesses a letter, it is displayed as one of the guessed letters:

word = w___ Guessed: w

The player starts with 9 lives, and when a player guesses a letter that is not in the word, the player loses a life:

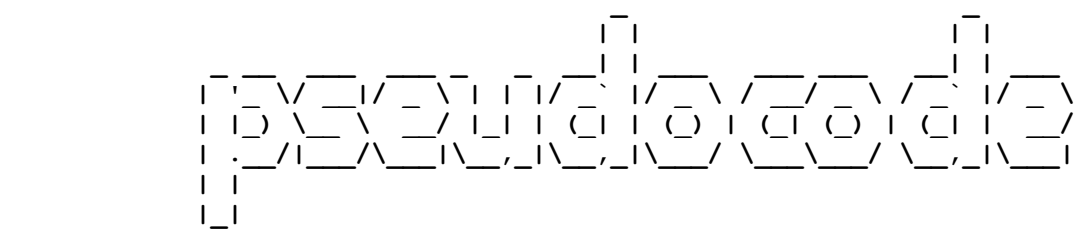
player guesses ‘x’

word = w___ Guessed: wx

Your guess was wrong. You have 8 lives left.

The player loses when they run out of lives.

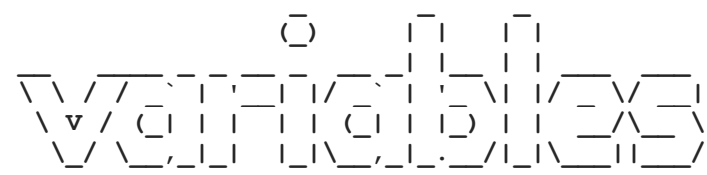
If the word has been fully uncovered, then the player wins!



```
int main(){
      //Main menu selection where player chooses One Player, Two
        Player, leaderboards, or quits game
      //When a game ends, player has option to play again or quit game
}

void onePlayer(){
      //Player chooses game mode and/or difficulty
      //Choose a random word from the word list text document
      //Display word
      //Run gameMech() function and return score
      //If player won, record player's name and score into leaderboards
      //Read every line from file into ldrbrd string
      //If there is less than 20 entries of HI-SCOREs in the file, then
        score places in the leaderboards. Or else, if there are 20
        entries, then compare the player's potentially new HI-SCORE with
        the ones in the file to check if it places in the leaderboards
      //Display score, and if a new HI-SCORE is set, then enter player's
        name and write it into file as a new entry
      //If 20 or more entries in leaderboard file, remove lowest score
        and write in the new entry at the end of the file. Else, write
        in the new entry at the end of the file.
}
```

```
void twoPlayer(){
      //Second player inputs word, clear screen, and display word's
        length
      //Run gameMech() function
}

int gameMech(string word){
      //GAME MECHANICS
      //Do While loops as long as player hasn't won or lost yet
      //It shows mystery word and player inputs guess with input
        validation.
      //If guess was incorrect, display deducted lives count
      //If lives equal 0, player loses and breaks out of function
      //If guess was correct, reveal letter in variable 'blnk'
      //After you win or lose, ask player if they want to replay or
        not
}

int find(string str, char c){
      //Check's to see if character is found inside a string. If it is,
      return its position. If not found, return -1 (0 would return as
      first char in string)
}

void sortString(string &str){
      //Sorts string in alphabetical order
```

```
}

void ldrbrd(){
      //Ask player to choose a difficulty to see leaderboard for and open
      appropriate file
      //Display names and scores of leaderboard in order of their scores
}

void selSort(){
      // Scan through each elements in the array
      //Save largest element found and its index from array
      //Swap largest element in array
}
```



```
unsigned int COL        //2D array's column size
```

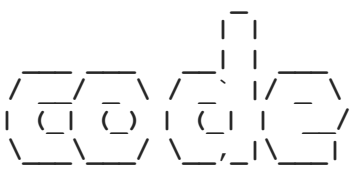
```
unsigned char select    //User's Main Menu input
unsigned char choice    //User's replay option
bool loop               //Is false if user wants to quit, otherwise
                          true
```

```
unsigned char sizeLst   //Size of word list
int scor                //Player's score
string diff             //Difficulty
string word             //Word to guess
string guessed          //Already guessed letters
unsigned seed           //Create seed for rand()
srand(seed)             //Seed the random number generator
fstream inDiff          //Read wordlist depending on difficulty chosen
fstream inLdbrd         //Input file for leaderboards
fstream outLdbrd        //Output file for leaderboards
Player plyr;            //Player's name and score
```

```
string word             //Word to guess
```

```
char guess              //Player's guess
string blnk             //Word with underscores for unknown letters
string guessed          //Already guessed letters
unsigned short lives    //Player's life
int mult                //Multiply scor each # of letters player
                        //uncovered in word
int streak              //Player's score
bool correct            //Flag for is player guessed a correct letter
bool win                //Flag for if player won or not
bool repeat             //If letter has already been guessed, false
```

```
                        otherwise
bool isOver             //Used as flag for game to keep looping
bool repeat             //Check if letter has been guessed


                ---------- int find() ----------
int i                   //Used to find position of char in string in
                          for loop

            ---------- void sortString() ----------
bool swap               //Used to break out of loop if a swap took
                          place
char temp               //Used to swap between two array elements
                         (temporary)

                ---------- void ldrbrd() ----------
const int SIZE          //Used to store leaderboard data
string data[SIZE][COL] //Store leaderboard names and scores as data
ifstream file           //Used to open leaderboards file
string diff             //Used to accept player's choice for difficulty

              ---------- void selSort() ----------
int largInd             //Used to store index of largest element of
                          array
int large               //Used to store value of largest element of
                          array
```



```
/*
 * Author: Kenzie Vasquez
 * Created on April 10, 2017, 4:22 PM
 * Purpose: Stores player's name and score for leaderboards
 */

#ifndef PLAYER_H
#define PLAYER_H

struct Player {
    unsigned short SIZE;
    string name;
    int scor;
};

#endif /* PLAYER_H */

//////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////


/* File:   Hangman – Project 1 – CIS-17A-42636
 * Author: Kenzie Vasquez
 * Created on April 10, 2017, 2:24 PM
 */
#include <cstdlib>    //For rand and srand, and atoi (ASCII to int)
#include <ctime>      //For time function
#include <fstream>    //File objects
#include <iostream>   //Input/Output objects
#include <string>     //String objects
#include <cstring>
using namespace std;  //Name-space used in the System Library


#include "Player.h"


const int COL = 2;
```

```
void onePlayer();          //Hangman game mode for one player
void twoPlayer();          //Hangman game mode for two players
Player gameMech(string);  //Hangman core game mechanics
int  find(string, char);   //Search for a char in a string var & return its pos
void sortString(string &); //Sort chars in str var for already guessed letters
void ldrbrd();             //Display top 20 players and their HI-SCOREs
void selSort(string [][COL], const int); //Sort leaderboard scores

int main(int argc, char** argv) {
    unsigned char slct,   //Main Menu option/selection
                  choice; //Play again option
    bool loop = true;     //Loops main menu

    while (loop){
        //Main menu selection with input validation
        //When a game ends, player has option to play again or quit game
        cout << " _                                                  \n"
               "| |                                                 \n"
               "| |__   __ _ _ __   __ _ _ __ ___   __ _ _ __       \n"
               "| '_ \\ / _` | '_ \\ / _` | '_ ` _ \\ / _` | '_  \\ \n"
               "| | | | (_| | | | | (_| | | | | | | (_| | | | |     \n"
               "|_| |_|\\__,_|_| |_|\\__, |_| |_| |_|\\__,_|_| |_|  \n"
               "                     __/ |                          \n"
               "                    |___/                           \n";

        cout << "\n   1. One Player!"
               "\n   2. Two Player!!"
               "\n   3. Leaderboards"
               "\n   X. Quit :( " << endl;

        do {
```

```
            cin >> slct; cin.ignore();//Player input
        } while(!(slct == '1' || slct == '2' || slct == 'X' || slct == 'x'
               || slct == '3'));

        switch (slct) {
            case '1': onePlayer(); break;
            case '2': twoPlayer(); break;
            case '3': ldrbrd();    break;
            case 'x': case 'X': loop = false;
        }

        if (loop) {
            cout << "Return to main menu? Y(1) or N(0)\n";
            do {
                cin >> choice; cin.ignore();
            } while (!(choice == 'y' || choice == 'Y' || choice == '1'
                    || choice == 'n' || choice == 'N' || choice == '0'));
        }

        system("clear"); //Clear screen

        if (choice == 'n' || choice == 'N' || choice == '0') {
            loop = false;
        }
    }

    cout << "Thanks for playing!\n";

    return 0;
}

//0000000111111111122222222223333333333444444444455555555556666666666777777777
```

```
//34567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
//****************************** onePlayer **********************************
//Purpose:  Function is to run the game as a one-player gamemode. onePlayer
//          reads random words from a text file depending on what difficulty the
//          player chose.
//
//Inputs:   Nothing is sent in           -> Description, Range, Units
//Output:   Nothing is returned out      -> Description, Range, Units
//*******************************************************************************
void onePlayer () {
    //Variable declaration
    unsigned char sizeLst= 0;  //Size of word list
    string diff,               //Difficulty
           word,               //Word used for hangman
           guessed = "",       //Already guessed letters
    //       name,             //Player's name for leaderboards
           *wordLst = nullptr;
    unsigned seed = time(0);   //Create seed from system time
    srand(seed);               //Seed the random number generator
    fstream inDiff,            //Input file for difficulty
            outLdbrd;          //Output file for leaderboards
    Player plyr;

    do {
        system("clear"); //clear screen

        //Player chooses difficulty with input validation
        do {
            cout << "Which difficulty would you like to play,\n"
                    "easy(1), moderate(2) or hard(3)?" << endl;
            cin >> diff; cin.ignore();
        } while (!(diff == "easy" || diff == "1" || diff == "moderate"
```

```
            || diff == "2" || diff == "hard" || diff == "3"));
    cout << endl;

    //Depending on player's choice of difficulty, load text file containing
    //custom # of words. If file couldn't load, return to difficulty menu.
    switch (diff[0]){
        case 'e': case '1':
            diff = "easy.txt";
            sizeLst = 101; break;
        case 'm': case '2':
            diff = "moderate.txt";
            sizeLst = 200; break;
        case 'h': case '3':
            diff = "hard.txt";
            sizeLst = 100; break;
        default: cout << "Error." << endl;
    }

    inDiff.open(diff, ios::in);

    wordLst = new string[sizeLst];

    //If file can't load, return to difficulty menu
    if (!inDiff.fail()) {
        for (int i = 0; i < sizeLst; i++){
            inDiff >> wordLst[i];
            //cout << wordLst[i] << endl;
        }
    } else cout << "File couldn't load. :(" << endl;
} while(inDiff.fail());

word = wordLst[rand() % sizeLst]; //Assign rand word from wordLst to
```

```
                                    //variable 'word'

system("clear"); //clear screen

//cout << "sizeLst: " << static_cast<unsigned>(sizeLst) << endl;
delete [] wordLst; //delete dynamic wordLst arr
//cout << "Word: " << word << endl;
cout << "LENGTH OF WORD: " << word.length() << "\n\n";

plyr = gameMech(word);

if (plyr.scor){
    int SIZE = 20,
        largInd;
    bool isNewScor = false;
    int count = 0;
    diff = "leaderboards-" + diff;
    fstream inLdrbrd(diff, ios::in);

    string ldrbrd[SIZE] = {};

    //Read every line from file into ldrbrd string
    for (int i = 0; i < SIZE; i++){
        string data;

        if (getline(inLdrbrd, data)){
            ldrbrd[i] = data;
            count++;
        }
    }

    //If there is less than 20 entries of HI-SCOREs in the file, then the
```

```
        //score places in the leaderboards. Or else, if there are 20 entries,
        //then compare the player's potentially new HI-SCORE with the ones in
        //the file to check if it places in the leaderboards.
        if (count < 20){
            isNewScor = true;
        } else {
            int scores[SIZE] = {};
            largInd = 0;

            for (int i = 0; i < SIZE; i++){
                scores[i] = atoi(ldrbrd[i].substr(10, ldrbrd[i].length() -
10).c_str());
                if (i && scores[i] < scores[i - 1]) largInd = i;
            }

            //isNewScore is flag
            if (plyr.scor > scores[largInd]) isNewScor = true;
        }

        cout << "Your score was " << plyr.scor << endl; //display score

        //If a new HI-SCORE is set, then enter player's name and write it into
        //the file as a new entry
        if (isNewScor){
            bool isValid = false; //Used to check if name is valid

            do {
                cout << "Enter your name of up to 10 characters: ";
                getline(cin, plyr.name);

                for (int i = 0; i < plyr.name.length(); i++){
                    if (plyr.name[i] != ' ') isValid = true;
                }
```

```
            } while(!isValid);

            while (plyr.name.length() < 10) plyr.name += " ";
            while (plyr.name.length() > 10) plyr.name.erase(plyr.name.length() - 1,
1);

            //create cstrings for the player's name and score
            string dig = "";
            int count = 0;
            char cstrN[plyr.name.length()] = {};
               //cstrS[plyr.scor.length()] = {};
            dig = to_string(plyr.scor);
            for(int i = 0; i < dig.length(); i++){
                if(isdigit(dig[i])) count++;
            }
            char cstrS[count] = {};
            strncat(cstrN, plyr.name.c_str(), plyr.name.length());
            strncat(cstrS, dig.c_str(), count);

            //If 20 or more entries in leaderboard file, remove lowest score
            //and write in the new entry at the end of the file.
            //Else, write in the new entry at the end of the file.
            char newLine = '\n';
            outLdbrd.open(diff, ios::out|ios::binary|ios::app);

            if (count >= 20){
                for (int i = 0; i < SIZE; i++){
                    //if (i != largInd) outLdbrd << ldrbrd[i] << endl;
                    char ldrbrdC[ldrbrd[i].length() + 1] = {};
                    if (i != largInd){
                        outLdbrd.write(ldrbrdC, sizeof(ldrbrdC));
                        outLdbrd.write(&newLine, sizeof(newLine));
                    }
```

```
                }
                //outLdbrd << plyr.name << plyr.scor << endl;
                outLdbrd.write(cstrN, sizeof(cstrN));
                outLdbrd.write(cstrS, sizeof(cstrS));
                outLdbrd.write(&newLine, sizeof(newLine));
            } else {
                //outLdbrd.open(diff, fstream::app);
                //outLdbrd << plyr.name << plyr.scor << endl;
                outLdbrd.write(cstrN, sizeof(cstrN));
                outLdbrd.write(cstrS, sizeof(cstrS));
                outLdbrd.write(&newLine, sizeof(newLine));
            }
        }

        outLdbrd.close(); outLdbrd.clear();
        inDiff.close(); inDiff.clear();
    }
}

//00000001111111111222222222233333333334444444444555555555566666666667777777778
//34567890123456789012345678901234567890123456789012345678901234567890123456789
//******************************** twoPlayer ***********************************
//Purpose:  Player Two inputs word for Player One to guess, lowercase all
//          letters in word, and then begin main game mechanics function.
//
//Inputs:   Nothing is sent in           -> Description, Range, Units
//Output:   Nothing is returned out      -> Description, Range, Units
//******************************************************************************
void twoPlayer(){
    string word = ""; //Word used for hangman

    system("clear");
```

```
    //Player Two inputs word for Player One to guess
    cout << "\nInput the word, Player Two! ";
    cin >> word; cin.ignore();

    //Converts user's capitalized word to all lowercase
    for (int i=0; i < word.length(); i++) word[i] = tolower(word[i]);

    system("clear"); //Clear screen

    //cout << "Word: " << word << endl;
    cout << "Length of word: " << word.length() << "\n\n";

    gameMech(word);
}

//00000001111111111222222222233333333334444444444555555555566666666667777777778
//34567890123456789012345678901234567890123456789012345678901234567890123456789
//*************************** gameMech ***************************************
//Purpose:  Loops as long as word hasn't been guessed OR player's live is not 0.
//          It shows mystery word and player inputs guess with input validation.
//          If guess was incorrect, display deducted lives count. After you win
//          or lose, ask player if they want to replay or not.
//
//Inputs:   string word  -> word for player to guess, Range, Units
//Output:   0, scor      -> Return points if player won, or none if they lost,
//                          -2147483648 to 2147483647, int
//*****************************************************************************
Player gameMech(string word){
    char guess;               //Player's guess
    string blnk = "",         //Blank lines for unknown letters
           guessed = "";      //Already guessed letters
```

```
unsigned short lives = 12; //Player's life
int mult,                  //Multiply scor each # of letters player
                           //uncovered in word
    streak = 1;            //Multiply scor for streaks of correct guesses
bool correct,              //Flag for is player guessed a correct letter
     win = true,           //Flag for if player won or not
     isOver = false,       //Loops as long as game isn't over
     repeat = false;       //Check if letter has been guessed
Player *plyr = new Player;
plyr->scor = 100;

//For every letter in word, add another underscore _ to variable 'blnk'
for (int i = 0; i < word.length(); i++) blnk += "_";

do {
    mult = 1;
    correct = true; //If guess was wrong, take a life away

    cout << blnk << "   Guessed: " << guessed << "   SCORE: " << plyr->scor
         << "\n";

    //Enter guess
    do {
        repeat = false;
        cout << "Guess: ";
        cin >> guess; cin.ignore();
        guess = tolower(guess); //make player's guess lowercase

        //If you guessed an already used letter
        if (find(guessed, guess) != -1){
            repeat = true;
            cout << "\nYou've already guessed this letter!\n";
```

```
            streak = 1;
        }

    //Validation - Check if guess is a lowercase letter that hasn't yet
    //been guessed
    } while ((guess < 96 || guess > 123) && repeat);

    //Only add player's guess in variable 'guessed' if it hasn't been
    //guessed yet
    if (find(guessed, guess) == -1){
        guessed += guess;
        sortString(guessed);
    }

    //If you didn't guess right, take a life away and lose 100 points
    for (int i = 0; i < word.length(); i++){
        if (find(word, guess) == -1){
            correct = false;
        }
    }

    if (correct == false){
        cout << "\nYour guess was wrong. You have "
             << --lives << " lives left!\n";
        plyr->scor = (plyr->scor >= 100 ? plyr->scor - 100 : 0); //Player loses
100 points
                                                  //from score
        streak = 1; //Reset streak multiplier to 1
    }

    //If you guessed right, increment scor multiplier and
    //increase streak multiplier
    for (int i = 0; i < word.length(); i++){
```

```
            if (guess == word[i]){
                blnk[i] = guess;
                mult++;
                streak += .5;
            }
        }

        plyr->scor += correct ? 100 * mult * streak : 0; //calculate score

        cout << (mult >= 4 ? "Woah!" : "") << endl;

        //If you win, break out of function
        if (word == blnk) {
            for (int i = 0; i < word.length(); i++) word[i] = toupper(word[i]);
            cout << "You won! The word was " << word << ".\n\n";
            win != win;
            isOver = true;
        }

        //If you lose, break out of function
        if (lives == 0){
            for (int i = 0; i < word.length(); i++) word[i] = toupper(word[i]);
            cout << "You lost! The word was " << word << ".\n\n";
            //return 0;
            win = true;
            isOver = true;
        }
    } while(!isOver);

    //Depending if you win or lose, return appropriate value
//    if (!win) return 0;
//    if (win) return plyr->scor + (lives * 50);
```

```
    return *plyr;
}

//0000000111111111122222222223333333333444444444455555555556666666666777777777778
//3456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
//********************************** find ************************************
//Purpose:  Sorts the characters of a string array in alphabetical order
//
//Inputs:   string guessed -> string of guessed chars, range & units varies,
//          char guess     -> player's guessed letter, -128 to 127 or 0 to 255, char
//Output:   int i, -1      -> position of char in string, 4 bytes
//****************************************************************************
int find(string str, char c){
    for (int i = 0; i < str.length(); i++) if (str[i] == c) return i;
    return -1;
}

//0000000111111111122222222223333333333444444444455555555556666666666777777777778
//3456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
//************************** sortString ************************************
//Purpose:  Sorts chars in a string var in alphabetical order.
//          (string variables were already converted to lowercase)
//
//Inputs:   string guessed (by reference) -> player's guessed letters, units vary
//Output:   Nothing is returned out       -> Description, Range, Units
//****************************************************************************
void sortString(string &str){
    bool swap;
    char temp;

    do {
        swap = false;
```

```
        for (int i = 0; i < (str.length() - 1); i++){
            if (str[i] > str[i + 1]){
                temp = str[i];
                str[i] = str[i + 1];
                str[i + 1] = temp;
                swap = true;
            }
        }
    } while (swap);
}

//0000000111111111122222222223333333333444444444455555555556666666666777777777778
//3456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
//****************************** ldrbrd ************************************
//Purpose:  Ask player which difficulty of leaderboard they want to see. Open
//          leaderboard according to choice, and display the player's place,
//          name, and score. on leaderboard, name, and score.
//
//Inputs:   Nothing is sent in       -> Description, Range, Units
//Output:   Nothing is returned out  -> Description, Range, Units
//*****************************************************************************
void ldrbrd(){
    const int SIZE = 20;
    string data[SIZE][COL] = {};
    ifstream file;
    string diff;

    system("clear");

    //open leaderboard file depending on what player wants to see
    do {
```

```
        cout << "Which difficulty of the leaderboards do you want to see,\n"
                "easy(1), moderate(2) or hard(3)? ";
        cin >> diff; cin.ignore();
    } while (!(diff == "easy" || diff == "1" || diff == "moderate"
            || diff == "2" || diff == "hard" || diff == "3"));
    cout << endl;

    switch (diff[0]){
        case 'e': case '1':
            file.open("leaderboards-easy.txt"); break;
        case 'm': case '2':
            file.open("leaderboards-moderate.txt"); break;
        case 'h': case '3':
            file.open("leaderboards-hard.txt");
    }

    system("clear");

    if (file){
        cout << " _                _           _                         _
\n"
                "| |              | |         | |                       | |
\n"
                "| | ___  __ _  __| | ___ _ __| |__   ___   __ _ _ __ __| |___
\n"
                "| |/ _ \\/ _` |/ _` |/ _ \\ '__| '_ \\ / _ \\ / _` | '__/ _` / __|
\n"
                "| |  __/ (_| | (_| |  __/ |  | |_) | (_) | (_| | | | (_| \\__ \\
\n"
                "|_|\\___|\\__,_|\\__,_|\\___|_|  |_.__/ \\___/ \\__,_|_|
\\__,_|___/ \n" << endl;

        cout << " #  NAME         HI-SCORE\n"
                "-------------------------- " << endl;
```

```
        //For loop is used for reading all the lines from the file, and
        //saving data (names and scores) into a 2D array
        for (int i = 0; i < SIZE; i++){
            string line;
            if(getline(file, line)){
                data[i][0] = line.substr(0, 10);
                data[i][1] = line.substr(10, line.length() - 10);
            }
        }
    } else cout << "There are no HI-SCOREs.\n";

    selSort(data, SIZE);

    //Display players' leaderboard position, name, and HI-SCORE
    for (int i = 0; i < SIZE; i++){
        if (data[i][0].length()){
            cout << (i < 9 ? " " : "") << i + 1 << "  " << data[i][0]
                 << "     " << data[i][1] << endl;
        }
    }

    cout << endl;
}

//0000000111111111122222222223333333333444444444455555555556666666666777777777
//3456789012345678901234567890123456789012345678901234567890123456789012345678
//****************************** selSort ************************************
//Purpose:  Selection sort modified from Gaddis book
//
//Inputs:   string data         -> player's name and score from leaderboards file,
//                                 Range varies, string
//          const int size = 20      -> Description, Range, Units
```

```
//Output:   Nothing is returned out  -> Description, Range, Units
//*************************************************************************
void selSort(string arr[][COL], const int size = 20){
    int largInd;
    int large;

    //Scan through elements in array
    for (int i = 0; i < (size - 1); i++){
        largInd = i;
        large = atoi(arr[i][1].c_str());

        //Save largest element found and its index from array
        for(int j = i + 1; j < size; j++){
            if (atoi(arr[j][1].c_str()) > large){
                large = atoi(arr[j][1].c_str());
                largInd = j;
            }
        }

        //swap elements in arrays
        for(int k = 0; k < COL; k++){
            string temp = arr[largInd][k]; //temp = a;
            arr[largInd][k] = arr[i][k];   //a    = b;
            arr[i][k] = temp;             //b    = temp;
        }
    }
}
```