



Solution: Mini Project 2

In this lesson, you will see the step by step solution review of the hangman game.

We'll cover the following




- Solution
 - Step 1: Initialize state variables
 - Step 2: Display game details
 - Step 3: Select secret word randomly
 - Step 4: Display blanks for every letter in a secret word
 - Step 5: Take a letter from the user and check whether the player guessed it right
 - Step 6: Display man on a gallow
 - Step 7: Player win/loss

Solution#

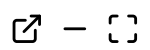
Press the **RUN** button and see the output!

If you want to try the game after some code changes, press the RUN button, then wait to be dropped back in the Terminal tab. Once there, kill the running program with Ctrl+c on Windows, or command+c on a Mac. Then, type the command `g++ main.cpp -o main` and hit enter to compile the program. If you don't get any compilation errors, you may run the program by typing `./main` and hitting enter.

 Search in directory...

/
main.cpp

```
76  "
77  "
78  "
79  "
80  "
81  "
82
83  cout << "The purpose of this game is
84  cout << "You have to guess one lette
85  cout << "Enter a lower-case letter a
86  cout << "Let's play the game!\n\n";
87  }
88
89  string chooseSecretWord() {
90      srand(time(NULL));
91
92      string animals[] = {"puppy","turtle"
93                          "sheep","turkey","chicken","hors
94                          "elephant","leopard","hippopotam
95                          "hedgehog"
96  };
97
98  int randomIndex = (rand() % 20);
99  string word = animals[randomIndex];
100  return word;
101  }
102
103  void replaceDashes(char guessWord[], int
```



Terminal





Hangman game

Step 1: Initialize state variables#

First, we must initialize some variables to store the game statistics.

```
1    int maxTries = 5;
2    int remainingTries = 5;
3    char guessLetter;
4    string secretWord;
5    int secretWordLength;
```

- `maxTries` indicates how many times the player can guess the wrong letter.
- `remainingTries` keeps track of the number of wrong tries that the player has left.
- `guessLetter` stores the character guessed by the user.

Step 2: Display game details#

We call the `displayGameDetails(maxTries)` function to display some necessary details about the game. It just uses the `cout` statement to print some instructions on the terminal.

Step 3: Select secret word randomly



We call the `chooseSecretWord()` to make an array of words and select the secret word randomly from the given array. In this function, we do the following:

- First, we create an array of strings called `animals[]` and store all the secret words in it.
- To choose the word randomly from an array, we use the `rand()` and `srand()` functions defined in the `<stdlib>` library. We also use the `time` function defined in the `<ctime>` library. By using these built-in functions, we can select a random index from an array and then return the word at that index.
- `rand()` takes nothing in input and generates a random number from **0 to 2147483647**. `rand()` does not generate truly random numbers. Instead, it generates a pseudo-random number, which means each time you compile the program, you get the same sequence of numbers. We can use the modulus operator to scale the random numbers within the given range.
- To generate a different sequence of random numbers each time the program executes, we use the `srand()` function. `srand()` seeds the random number generator so that each time we start in a different place. `srand()` takes seed in its input. We only need to call `srand()` once in a program. The most common way to seed the random number is to use the current time in `srand()` input. We use the `time()` function defined in the `ctime` library to know the current time. The `time()` function takes `NULL` in its input.

```
1 #include<stdlib>
2
3 #include<ctime>
```



```
4
5 string chooseSecretWord() {
6
7     string animals[] =
8     {"puppy","turtle","rabbit",
9      "raccoon","kitten","hamster",
10     "sheep","turkey","chicken",
11     "horse","chimpanzee","kangaroo",
12     "koala", "elephant","leopard","hippopotamus",
13     "giraffe","crocodile","alligator",
14     "hedgehog"
15 };
16
17 srand(time(NULL));
18 int randomIndex = (rand() % 20);
19 string word = animals[randomIndex];
20 return word;
21 }
```



Step 4: Display blanks for every letter in a secret word#

Now that we have selected the secret word, we want to display the number of letters present in the word by using the blanks _ .

First, we find the length of that secret word by using `.length()` property of the string. We initialize an array of characters `guessWord` whose length is equal to the length of the secret word. Initially, we fill all the array letters with the blanks – in function `replaceDashes` and use the `displayWord()` function to print its letters on the console.

```
1     secretWordLength = secretWord.length();
2
3     char guessWord[secretWordLength];
4     replaceDashes(guessWord, secretWordLength);
5     cout << "Your guess word is:";
6     displayWord(guessWord, secretWordLength);
```





Step 5: Take a letter from the user and check whether the player guessed it right#

We need to check the following conditions:

- If the player guesses a correct letter, that is present in the secret word. Then we have to reveal that letter in the correct position.
- If the user guesses a letter that has already been revealed before, then do nothing.
- If the user guesses the wrong letter, deduct the number of remaining attempts.

To implement this logic, we keep executing the loop statements until the user guesses the correct word or until a certain number of incorrect guesses is reached. We take input from the user and then call the `isGuessTrue()` function to check the above-mentioned conditions.

```
1  int isGuessTrue(string secretWord, char guessWord[], char letter) {  
2      int flag = 0;  
3      for (int i = 0; i < secretWord.length(); i++) {  
4          if (secretWord[i] == letter) {  
5              if (guessWord[i] == secretWord[i]) {  
6                  flag = 2;  
7              } else {  
8                  guessWord[i] = secretWord[i];  
9                  flag = 1;  
10             }  
11         }  
12     }  
13     return flag;  
14 }
```

- If the `isGuess()` function **returns 0**, it means the player has guessed the wrong letter and our remaining tries are subtracted by one. We also display some parts of the man on a gallows by calling the `displayMan()` function.
- If the player has guessed the correct letter, the `isGuess()` function reveals the word at that index and **returns 1** to the calling point.
- If the `isGuess()` function **returns 2**, it means the letter has already been revealed before.

```

1  int guess = isGuessTrue(secretWord, guessWord, guessLetter);
2
3      if (guess == 0) {
4          remainingTries--;
5          cout << "\nWhoops! that letter is not present in the word" << endl;
6          displayMan(remainingTries);
7      }
8      if (guess == 1) {
9          cout << "\nYay! You have found the letter" << endl;
10     }
11     if (guess == 2) {
12         cout << "\nYou have already guessed this letter. Try something else" << endl;
13     }
14
15 }
```

Step 6: Display man on a gallow#

We need to draw a man on the gallow in five different steps. For this, we call the `displayMan` function, which takes the number of remaining wrong tries that a player can have in its input and uses the `switch()` statement along with `cout` to display the hangman accordingly.

≡  (/learn)

```

1  void displayMan(int remainingGuess) {
```

```

2
3     string part[4];
4     switch (remainingGuess) {
5     case 0:
6         part[3] = "|";
7     case 1:
8         part[2] = "/|\\\\";
9     case 2:
10        part[1] = "/|\\\\";
11    case 3:
12        part[0] = "( )";
13        break;
14    }
15
16    cout << "-----\\n";
17    cout << " |          " << part[3] << endl;
18    cout << " |          " << part[3] << endl;
19    cout << " |          " << part[0] << endl;
20    cout << " |          " << part[1] << endl;
21    cout << " |          " << part[2] << endl;
22    cout << " |\\n"
23    cout << " |\\n"
24    cout << "-----\\n";
25 }

```



Step 7: Player win/loss#

While iterating through the loop statements, we keep checking whether the guess word is equal to the secret word at the end. If yes, we return 0 to the calling point to terminate the application.

```

1  if (secretWord == guessWord) {
2      cout << "\\nCongratulation! You won." << endl;
3      return 0;
4  }

```



If we come out of the loop and the secret word is still not equal to the guess word, the player loses the game!!



```
1  if (secretWord != guessWord){  
2      cout << "\nToo many Guesses! You have been hanged." << endl;  
3      cout << "\nThe secret word was: ";  
4      displayWord(secretWord, secretWordLength);  
5  }
```



You can play around with the code and make any changes to it to understand things better.

The next chapter contains the remarks for this course.

[← Back](#)

[Next →](#)

[Solution: Mini Project 1](#)

[Introduction to Structures](#)

☒ [Mark as Completed](#)



[Report an Issue](#)