Hangman

The word to guess, selected randomly by the program, is represented by a row of dashes, representing each letter of the word. If the guessing player suggests a letter which occurs in the word, then the program should reveal all in their correct positions. If the suggested letter does not occur in the word, it counts as a miss. On the tenth miss the player loses. For example, if the word is `cat`, the guesses are `e` and `a`, then the player should see: `_a_ misses: e`. The program should:

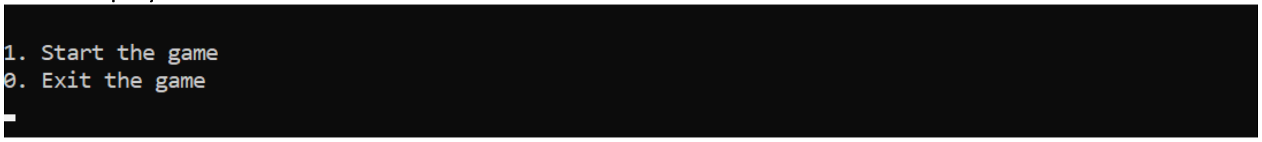
- Select a random word (of length given by the user) from the list.
- Notify the user if the guessed letter has already been guessed.
- Let the user change difficulty: change the allowed number of misses.

As required above, we need main three variables that is menu of type char to instruct and give information to User so to get inputs from him/her, and a variable of type integer that can store number of misses to track the User's "life-length" and a variable to get User's guesses to find a word which by intuition would require it to be of type char.

Two menus will be clever to choose. And both menus will contain choices that is to be made by User. For simplicity, the choices will be numbered, therefore, User must type only digits e.g: 0 to exit the program or 0 to start the game. For this reason, a variable of type integer will be needed under name "choice".

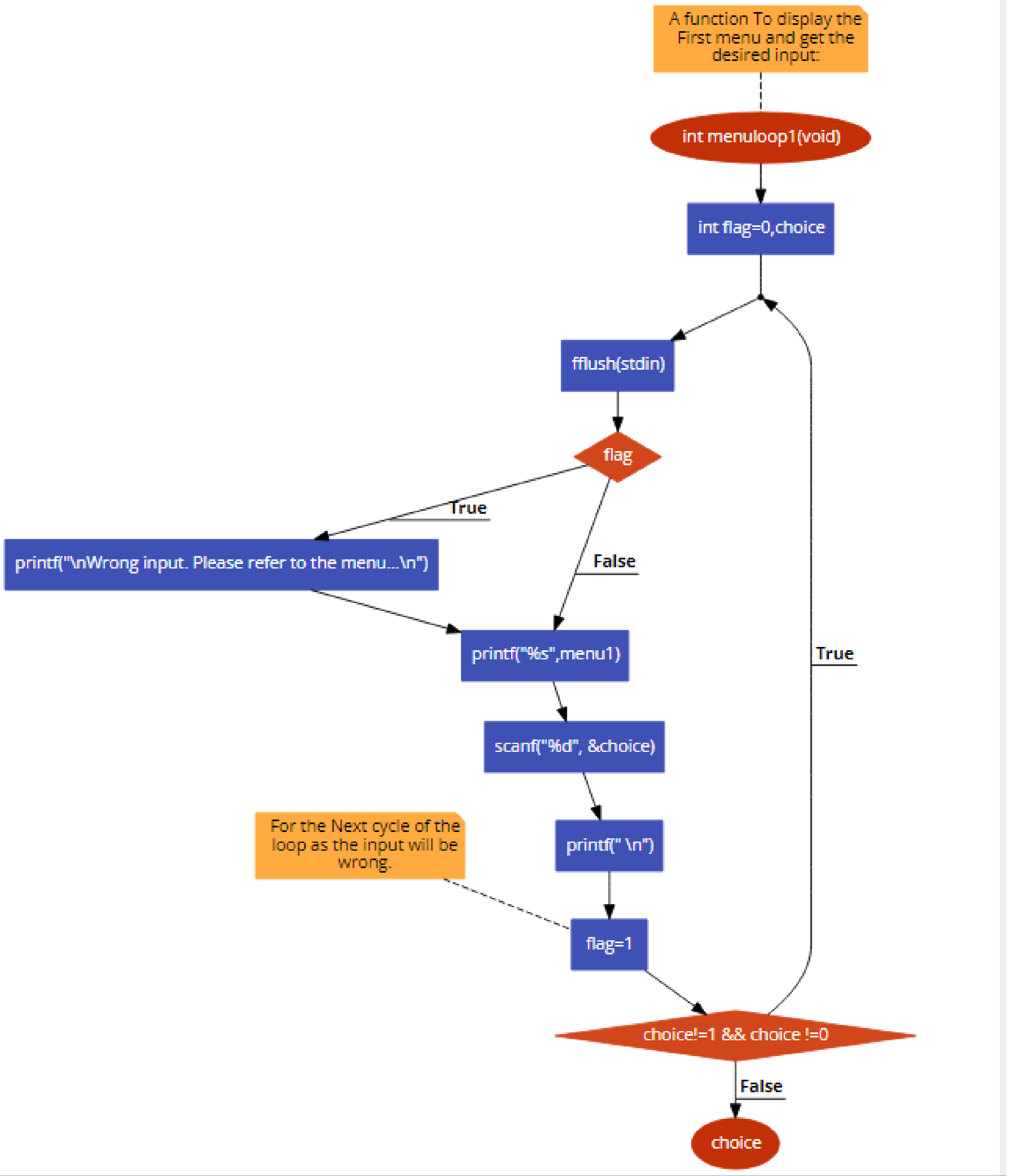
For handling 1st menu there will be a function for it named "menuloop1" with return type integer and with no parameters.

It will display the first menu illustrated below and asks the user to choose



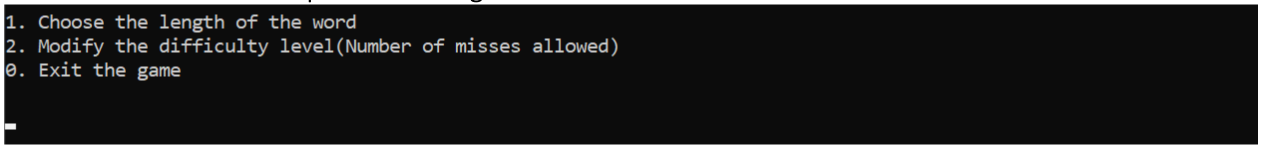
```
1. Start the game
0. Exit the game
```

In this function, a do-while loop is suitable to use as we want to keep asking the user until the valid input which is either 0 or 1. As the flowchart below illustrates the loop will keep running until valid input. As the input is inserted and the condition for it is satisfied the loop will terminate and the function will immediately return the choice number with the help of local variable "choice". Integer variable "flag" will be used to set If statement false so to not display the "wrong input" message for the first cycle. For the second cycle it will be set 1 because when the loop goes for the second cycle it already will mean the input was wrong. Eventually, it will return choice and we step into main function.


A function To display the First menu and get the desired input:
int menuloop1(void)
int flag=0,choice
fflush(stdin)
flag
True
printf("\nWrong input. Please refer to the menu...\n")
False
printf("%s",menu1)
scanf("%d", &choice)
printf(" \n")
For the Next cycle of the loop as the input will be wrong.
flag=1
True
choice!=1 && choice !=0
False
choice

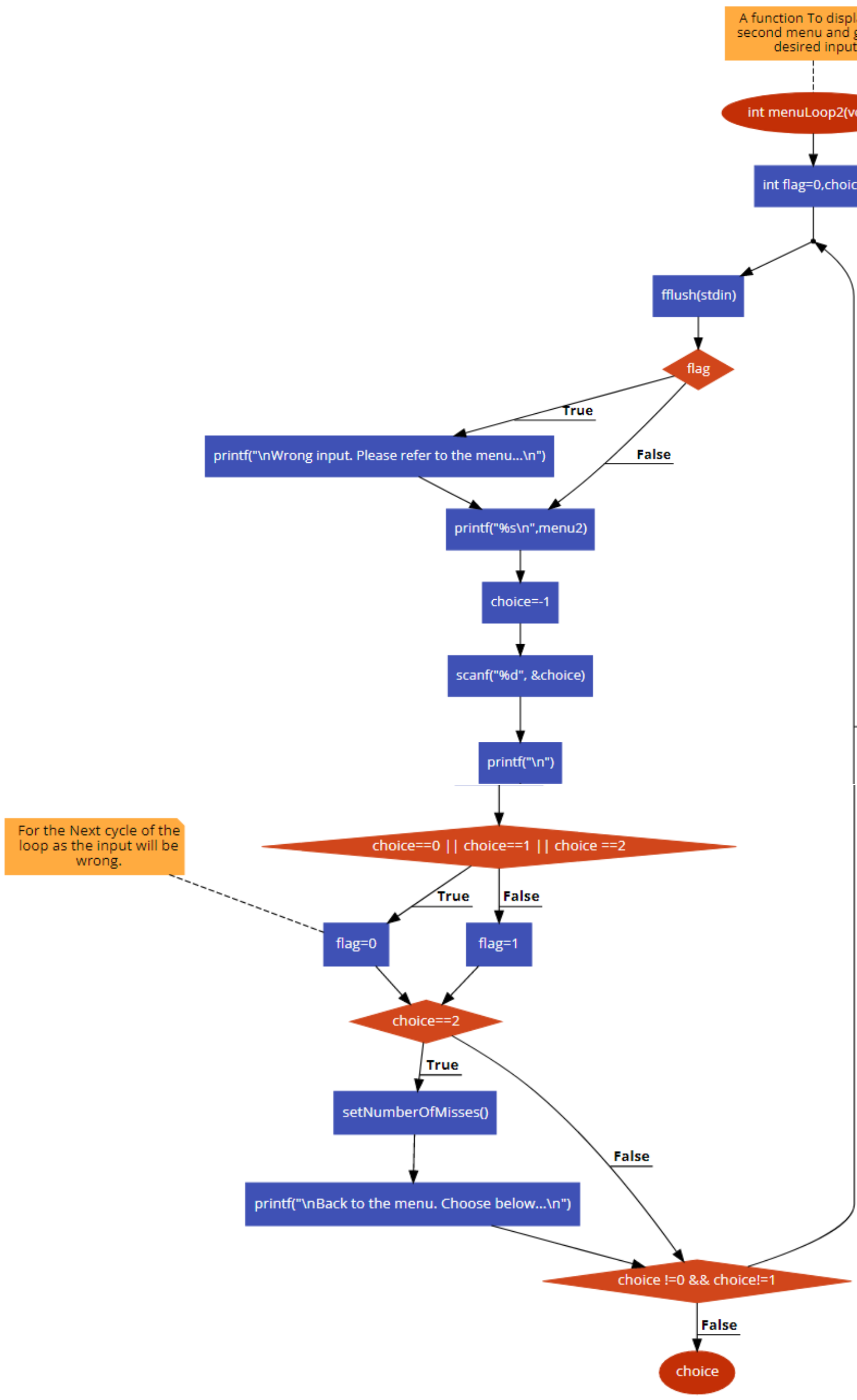
1.Flowchart for the “menuloop1” function.

In main function, we decide to exit the program if 0 was pressed. Otherwise, main will call the next function called “menuloop2” for dealing with the second menu like below



```
1. Choose the length of the word
2. Modify the difficulty level(Number of misses allowed)
0. Exit the game

_
```

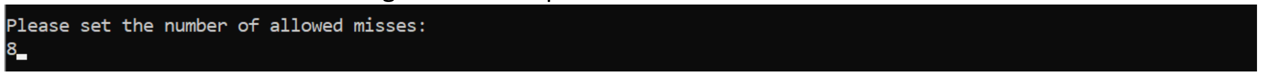
This function's concern is to get a valid choice for choosing the length of the word, modifying the difficulty level and for exiting the game.


A function To display the second menu and get the desired input:
int menuLoop2(void)
int flag=0,choice
fflush(stdin)
flag
True
False
printf("\nWrong input. Please refer to the menu...\n")
printf("%s\n",menu2)
choice=-1
scanf("%d", &choice)
printf("\n")
True
For the Next cycle of the loop as the input will be wrong.
choice==0 || choice==1 || choice ==2
True
False
flag=0
flag=1
choice==2
True
setNumberOfMisses()
False
printf("\nBack to the menu. Choose below...\n")
choice !=0 && choice!=1
False
choice

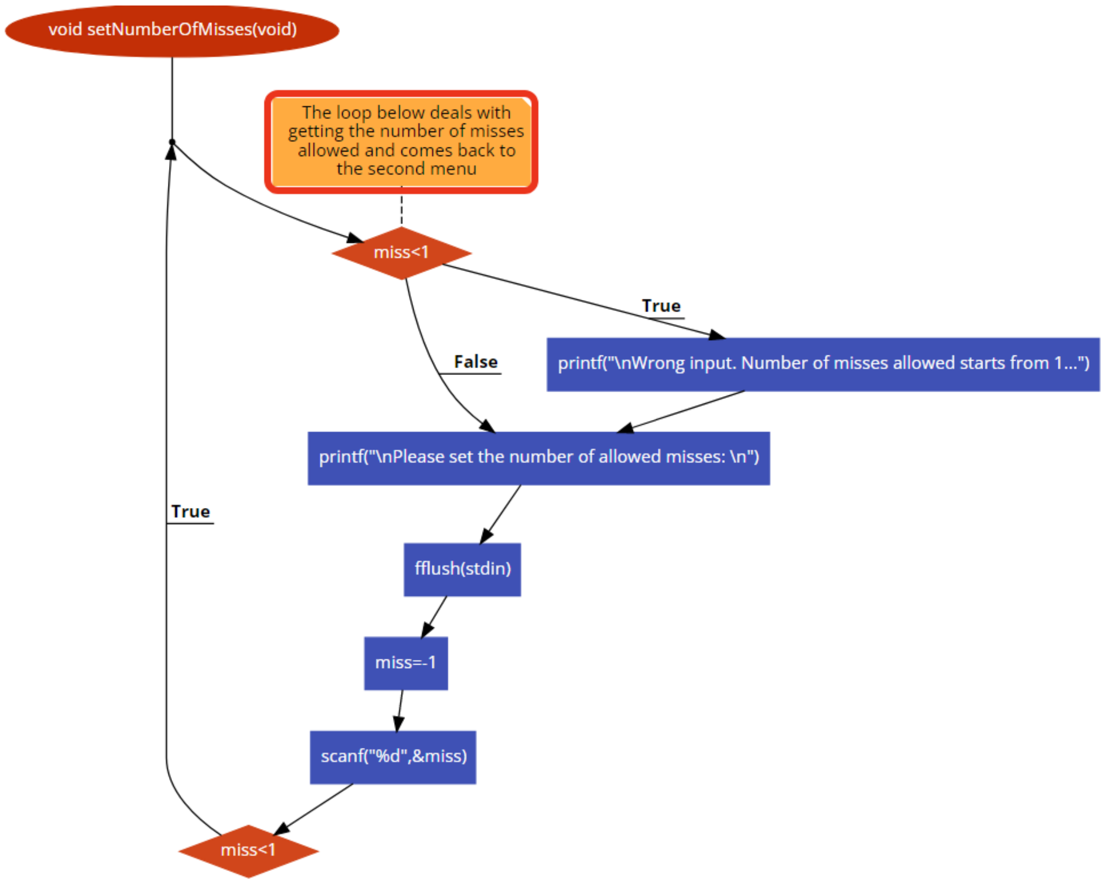
"flag" and "choice" local variables will be used the same way as in the previous function. The condition for the loop will be the same as "menuloop1" as well. Th reason for this is, in case choice is 2 we will need another function to set number of misses ("setNumberOfMisses") and we will do it inside the loop.

So, “setNumberOfMisses” will be called as many times as the user chooses choice 2 from the menu. And it will always get back to the 2nd menu. However, when either 0 or 1 is chosen we no longer need to go on to call another function but simply terminate the loop and return the choice.

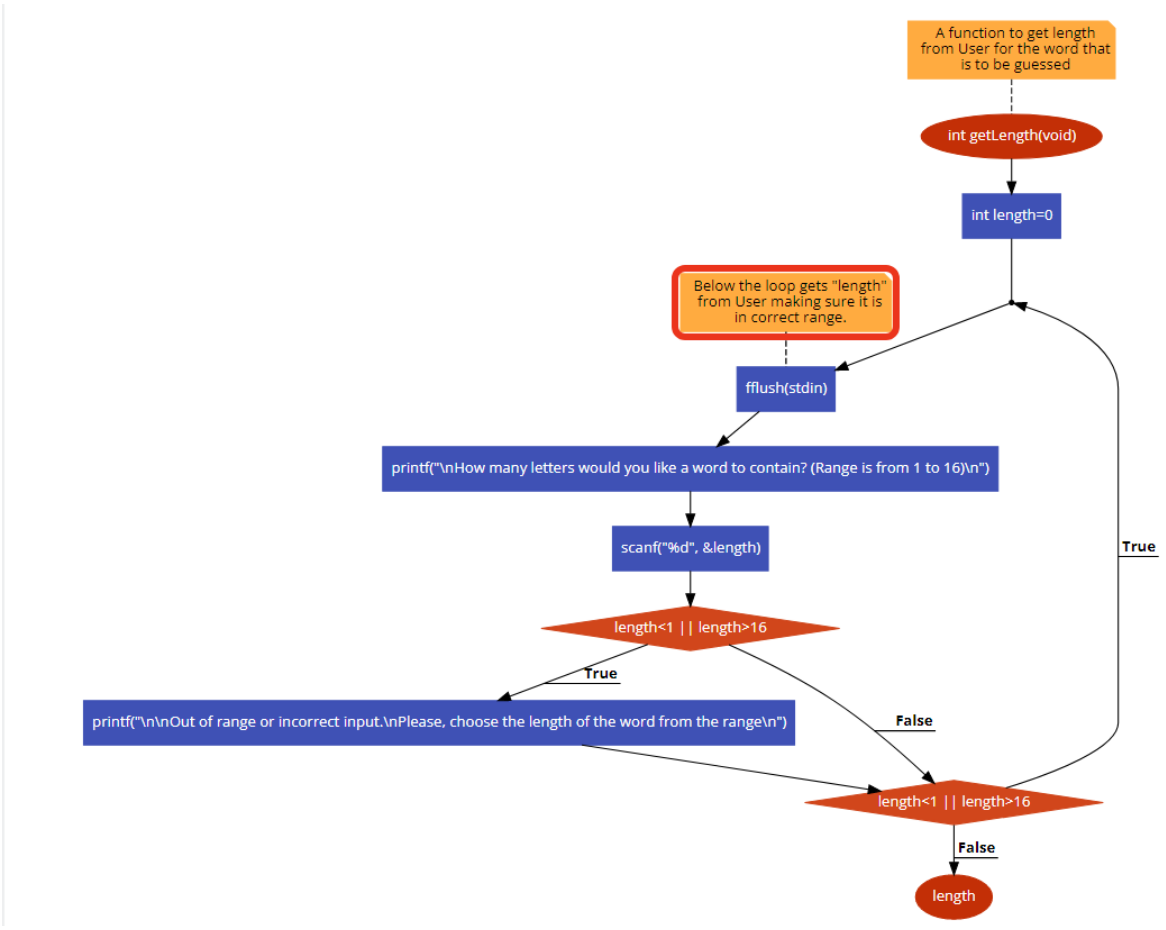
As number of misses allowed set by default the 2nd choice is optional. In case, the user wants a challenge or easy play he or she can choose it. After that “menuloop2” function will call “setNumberOfMisses” function to ask the user to change it and accept a valid modification.



```
Please set the number of allowed misses:
8
```


void setNumberOfMisses(void)
The loop below deals with getting the number of misses allowed and comes back to the second menu
miss<1
True
False
printf("\nWrong input. Number of misses allowed starts from 1...")
printf("\nPlease set the number of allowed misses: \n")
fflush(stdin)
miss=-1
scanf("%d",&miss)
True
miss<1

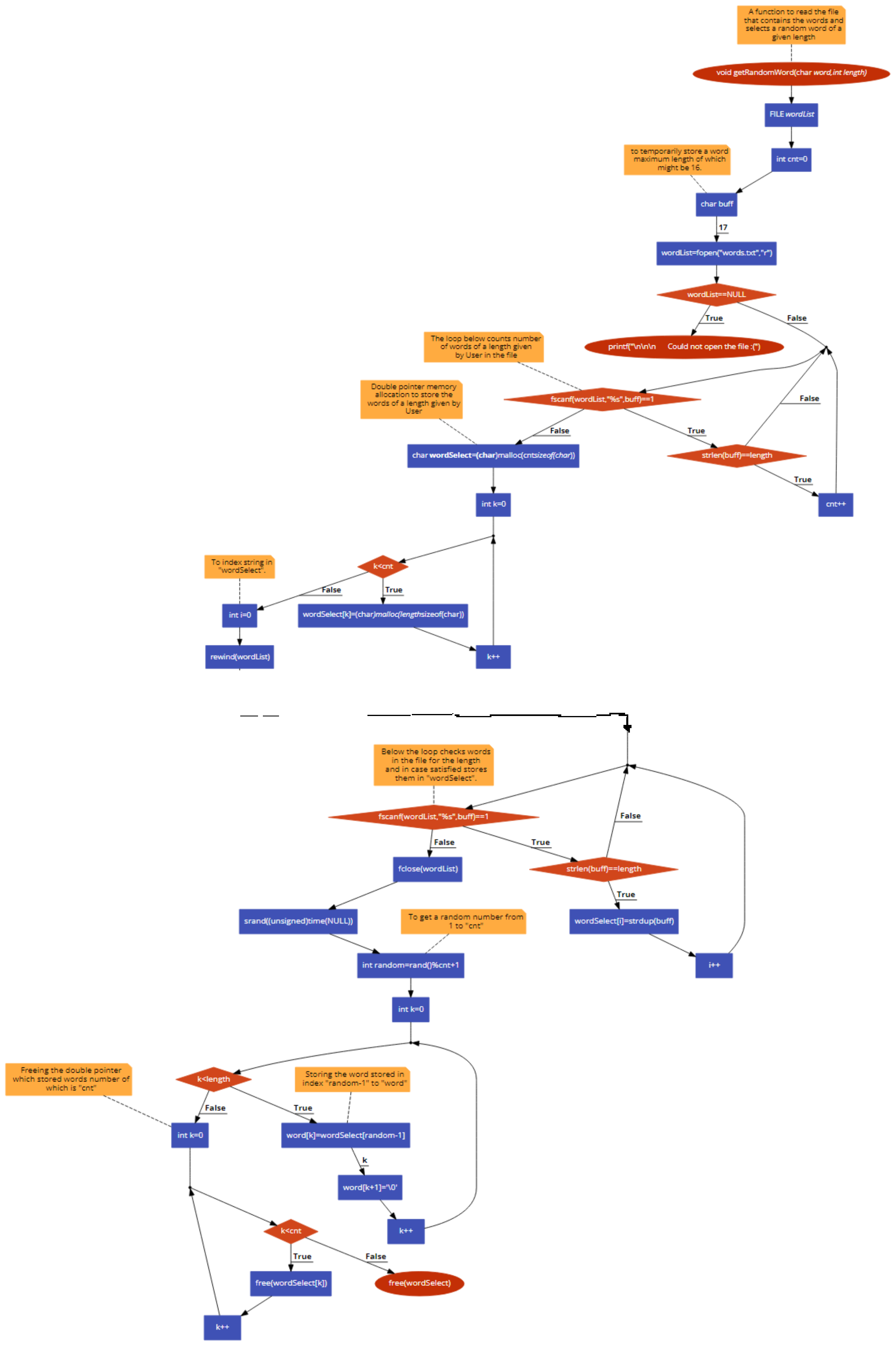
As a variable that stores number of misses allowed is global the function is void return type with no input parameters. After this function we will get back to “menuloop2” to display the 2nd menu again and choose. User can still modify it as long as he/she wants to.
After the choice has been made by User, main function either terminates the program or proceeds depending on the choice.
Proceeding means the user has chosen choice 1 which is choosing the length for a word. Main calls “getLength” function which will deal with making sure to get valid length for a word. “getLength” is of integer return type with no input parameter as we will assign this function’s output to the main function’s local variable.


A function to get length from User for the word that is to be guessed
int getLength(void)
int length=0
Below the loop gets "length" from User making sure it is in correct range.
fflush(stdin)
printf("\nHow many letters would you like a word to contain? (Range is from 1 to 16)\n")
scanf("%d", &length)
length<1 || length>16
True
printf("\n\nOut of range or incorrect input.\nPlease, choose the length of the word from the range\n")
False
length<1 || length>16
True
False
length

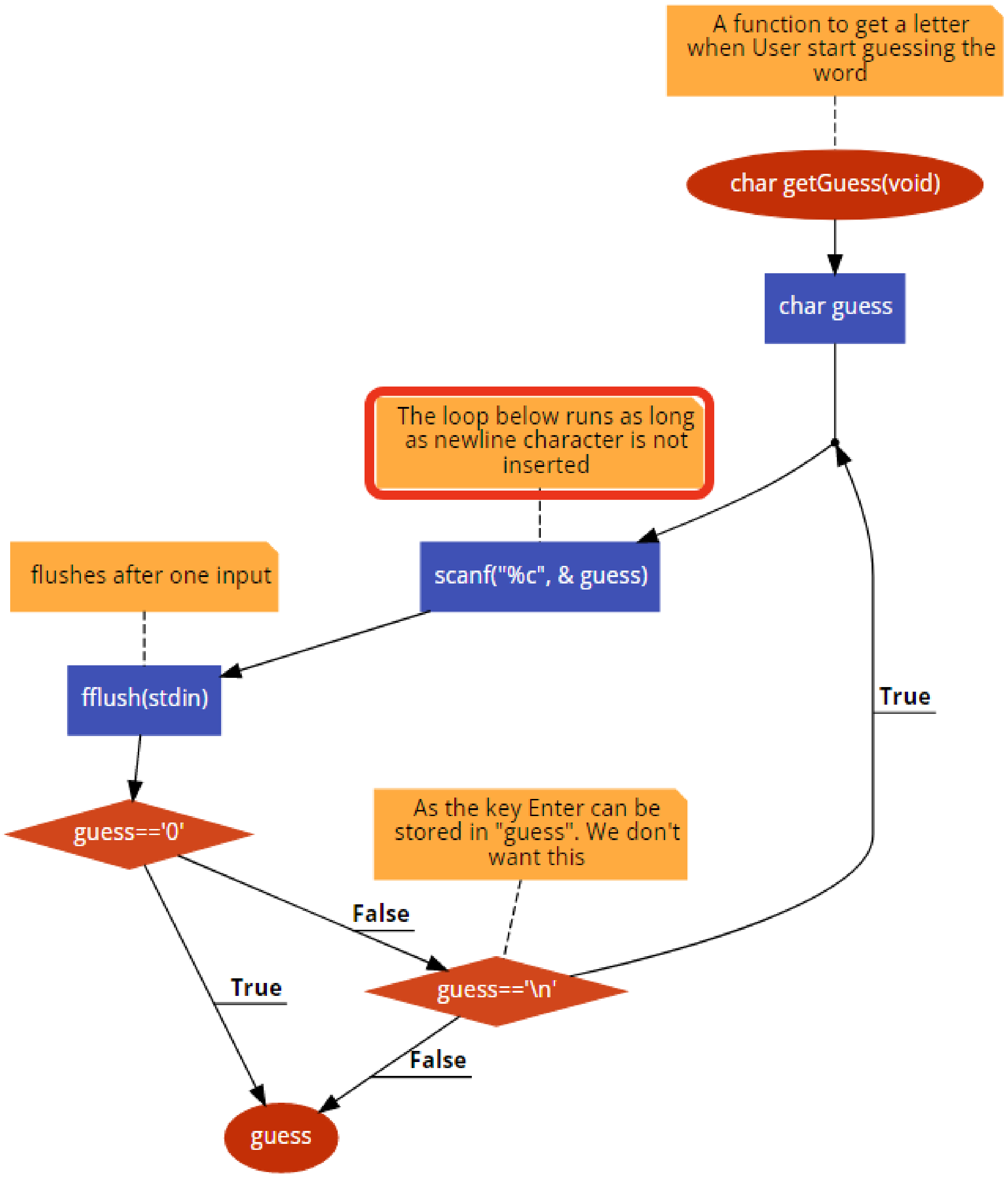
“getLength” function’s flowchart.

As soon as we have number of misses allowed and length for a word stored in “miss” and “length” variables of type integer respectively we are practically ready start Hangman. But before that, we must get a random word of length given by the User from a file. We declare a pointer of type char and we allocate memory for it with size “length”+1. We will need a function that can handle the file and the pointer passed as input parameter. We will name this function “getRandomWord”. The function will receive “length” and “word” as input parameter and will not return anything. The reason for this is, a random word will be stored in “word” so as the function changes its content it will be changed for main as well.

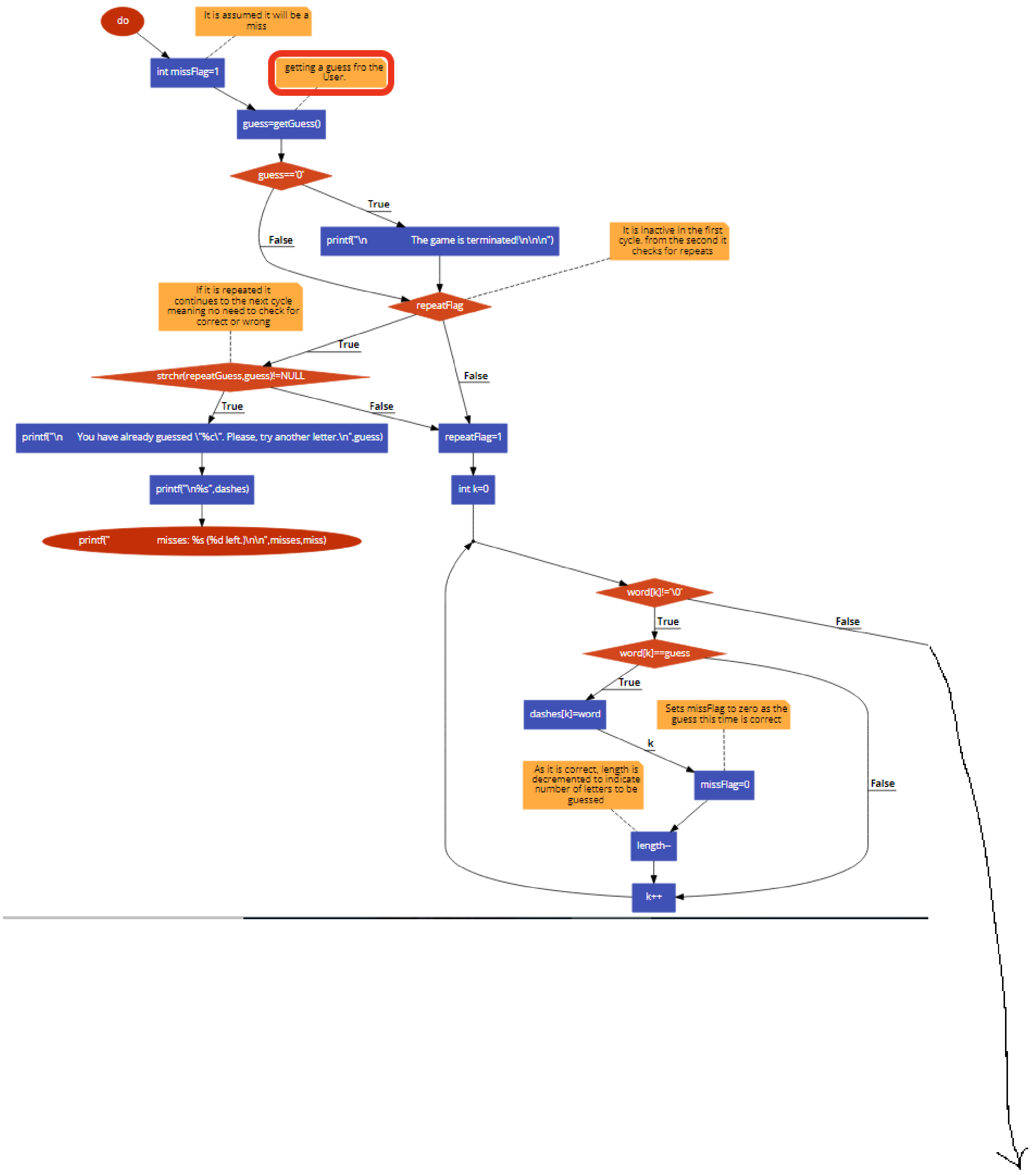
“getRandomWord” will open the file. If unsuccessful it will return with error message which will cause termination of the program in main. Otherwise, using loop we will count the words in the file that matches our “length”. As we get the count in “cnt” integer variable we will use it and “length” to declare double pointer “wordSelect” with memory allocation. Going forward, we will read the file one more time, but this time storing the words that equals our “length” in to “wordSelect”. Using srand() we get a random number and we will derive from “wordSelect” at “random” index by storing it into “word”. Eventually we close the file and free “wordSelect”.


A function to read the file that contains the words and selects a random word of a given length
void getRandomWord(char *word,int length)*
FILE *wordList*
int cnt=0
to temporarily store a word maximum length of which might be 16.
char buff
17
wordList=fopen("words.txt","r")
wordList==NULL
True
False
printf("\n\n\n Could not open the file :(")
The loop below counts number of words of a length given by User in the file
fscanf(wordList,"%s",buff)==1
Double pointer memory allocation to store the words of a length given by User
False
True
strlen(buff)==length
False
True
char **wordSelect=(char**)malloc(cnt*sizeof(char*))
cnt++
int k=0
To index string in "wordSelect".
k<cnt
False
True
int i=0
wordSelect[k]=(char*)malloc(length*sizeof(char))
rewind(wordList)
k++
Below the loop checks words in the file for the length and in case satisfied stores them in "wordSelect".
fscanf(wordList,"%s",buff)==1
False
False
True
fclose(wordList)
strlen(buff)==length
True
srand((unsigned)time(NULL))
To get a random number from 1 to "cnt"
wordSelect[i]=strdup(buff)
int random=rand()%cnt+1
i++
int k=0
Freeing the double pointer which stored words number of which is "cnt"
k<length
Storing the word stored in index "random-1" to "word"
False
True
int k=0
word[k]=wordSelect[random-1]
k
word[k+1]='\0'
k++
k<cnt
True
False
free(wordSelect[k])
free(wordSelect)
k++

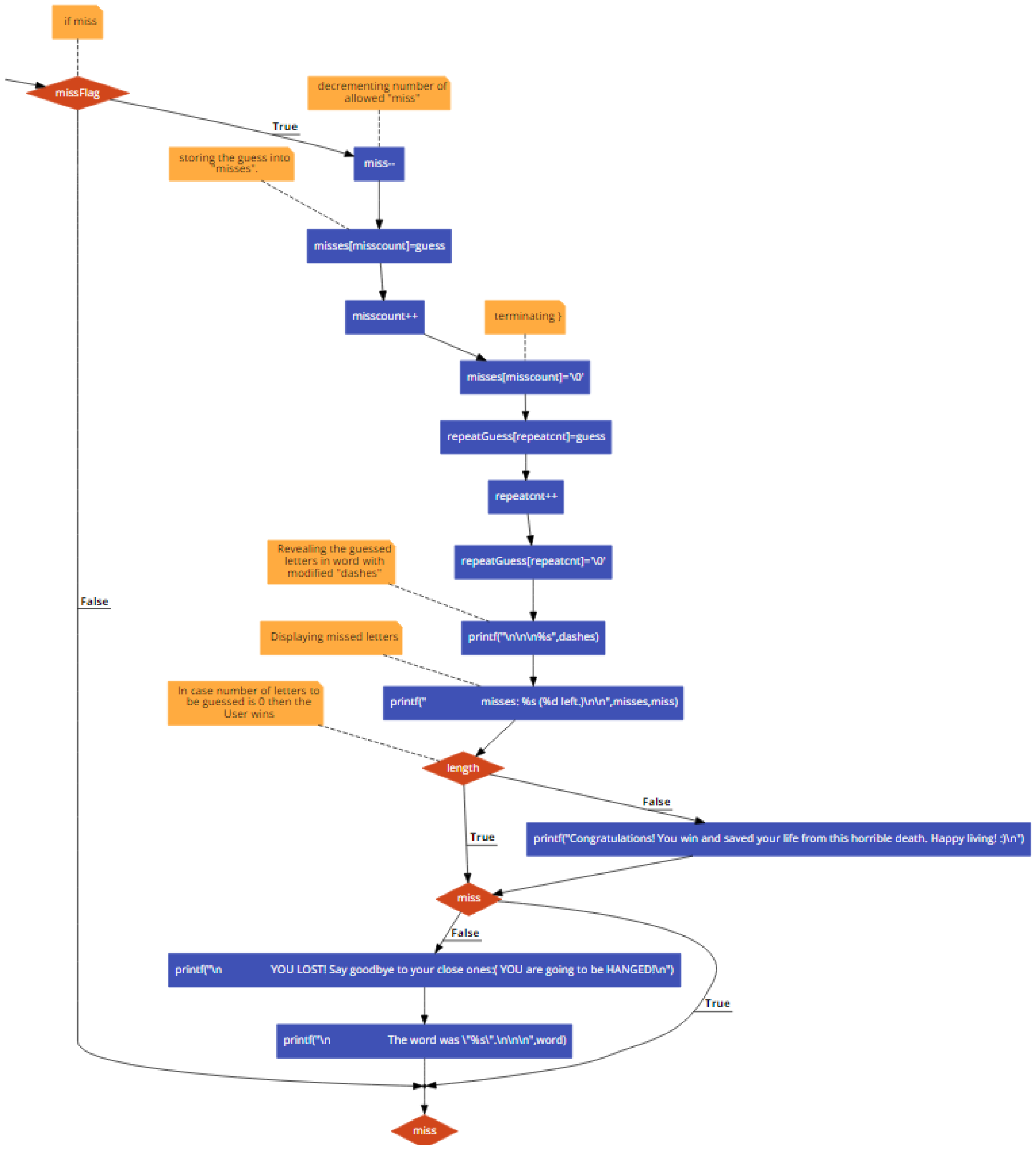
As we get the random word in “word”, we will declare three more dynamically allocated memory for three pointers: “dashes” with size “length”+1 to represent dashes and swap dashes with guessed letter if correct. “misses” with size “miss”+1 to display guessed letters that were wrong. “repeatGuess” with size “length”+ “miss” to store already guessed letters and warn the User about it if guess was repeated.
We fill “dashes” with dashes and at the end terminating ‘\0’, “miss” and “repeatGuess” with ‘\0’.
We will need to get a letter from the user. Therefore, for it, we create a function “getGuess” which interacts with the user asks for a guess and makes sure to get a valid one. It gets nothing as parameter, but returns char type. It will contain loop with condition to run as long as the input is newline character.


A function to get a letter when User start guessing the word
char getGuess(void)
char guess
The loop below runs as long as newline character is not inserted
scanf("%c", & guess)
flushes after one input
fflush(stdin)
guess=='0'
As the key Enter can be stored in "guess". We don't want this
False
guess=='\n'
True
True
False
guess

After getting the guess, we check for the repeat if so we continue for the next cycle, if not we proceed to check it in “word”. In case we find the character in the selected word we will reveal it in “dashes” at the respective index. If the guess is wrong we will count it as miss and store it in “misses”.
Each cycle will check for “miss” as soon as it reaches zero the user will lose and the loop will terminate.
Each cycle “length” will be decremented if the guess was correct and in case it reaches zero first the user will win.



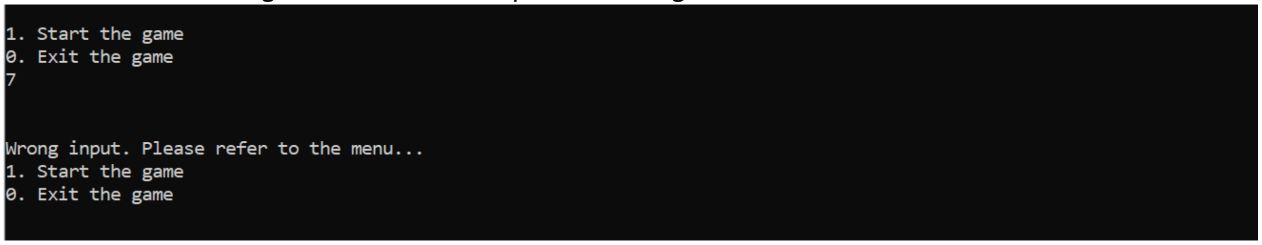
do
It is assumed it will be a miss
int missFlag=1
getting a guess fro the User.
guess=getGuess()
guess=='0'
True
False
printf("\n The game is terminated!\n\n\n")
It is inactive in the first cycle. from the second it checks for repeats
repeatFlag
If it is repeated it continues to the next cycle meaning no need to check for correct or wrong
True
False
strchr(repeatGuess,guess)!=NULL
True
False
printf("\n You have already guessed \"%c\". Please, try another letter.\n",guess)
repeatFlag=1
printf("\n%s",dashes)
int k=0
printf(" misses: %s (%d left.)\n\n",misses,miss)
word[k]!='\0'
True
False
word[k]==guess
True
dashes[k]=word
Sets missFlag to zero as the guess this time is correct
k
missFlag=0
False
As it is correct, length is decremented to indicate number of letters to be guessed
length--
k++


if miss
missFlag
True
decrementing number of allowed "miss"
miss--
storing the guess into "misses".
misses[misscount]=guess
misscount++
terminating }
misses[misscount]='\0'
repeatGuess[repeatcnt]=guess
repeatcnt++
repeatGuess[repeatcnt]='\0'
Revealing the guessed letters in word with modified "dashes"
printf("\n\n\n%s",dashes)
Displaying missed letters
printf("   misses: %s (%d left.)\n\n",misses,miss)
In case number of letters to be guessed is 0 then the User wins
False
length
False
printf("Congratulations! You win and saved your life from this horrible death. Happy living! :)\n")
True
miss
False
printf("\n   YOU LOST! Say goodbye to your close ones:(YOU are going to be HANGED!\n")
True
printf("\n   The word was \"%s\".\n\n\n",word)
miss

After that, we will free all allocated memories and terminate the program.

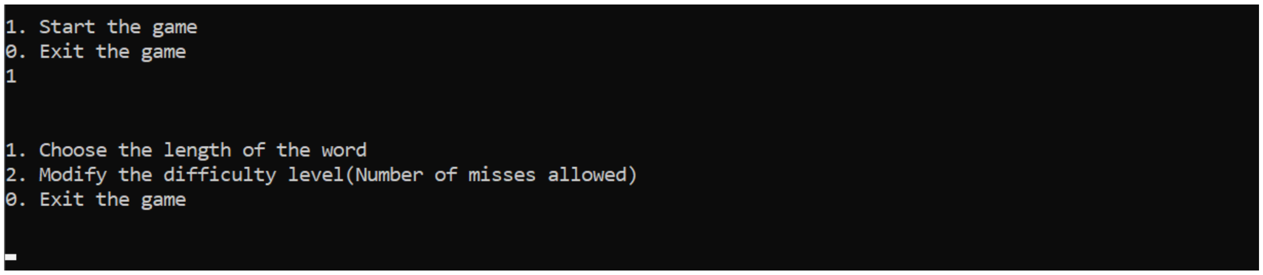
Testing

As the first interaction happens in "menuloop1" function, we tested to give wrong input. As expected, the result was warning about the invalid input and asking the user to refer to the menu.



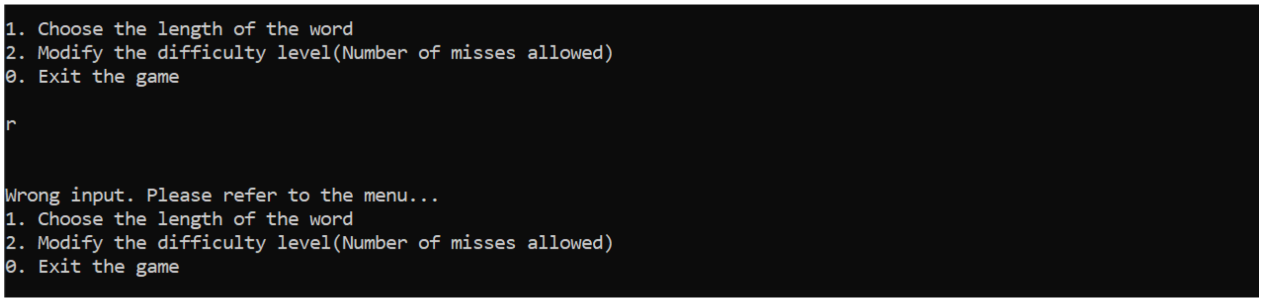
```
1. Start the game
0. Exit the game
7


Wrong input. Please refer to the menu...
1. Start the game
0. Exit the game
```

After we have given 1, the program proceeded to display the 2nd menu



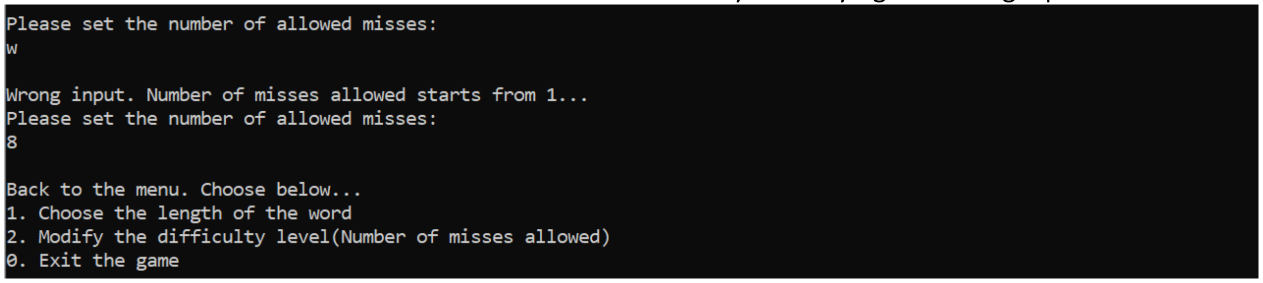
```
1. Start the game
0. Exit the game
1


1. Choose the length of the word
2. Modify the difficulty level(Number of misses allowed)
0. Exit the game
```

We deliberately gave wrong input as character type and this time as well we got warning message.



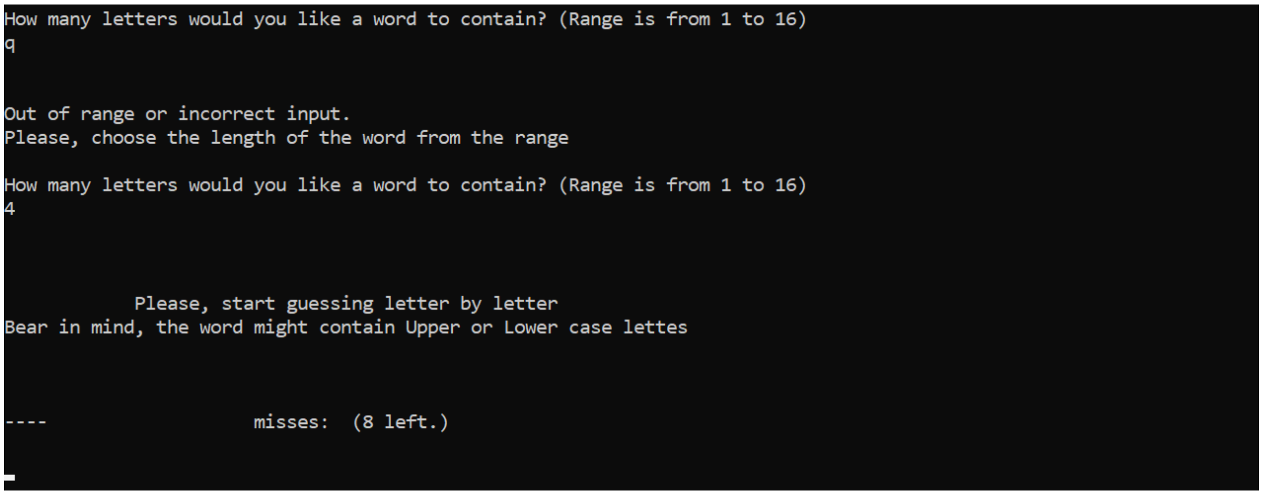
```
1. Choose the length of the word
2. Modify the difficulty level(Number of misses allowed)
0. Exit the game

r


Wrong input. Please refer to the menu...
1. Choose the length of the word
2. Modify the difficulty level(Number of misses allowed)
0. Exit the game
```

Below we modified the number of misses allowed successfully after trying out wrong input.



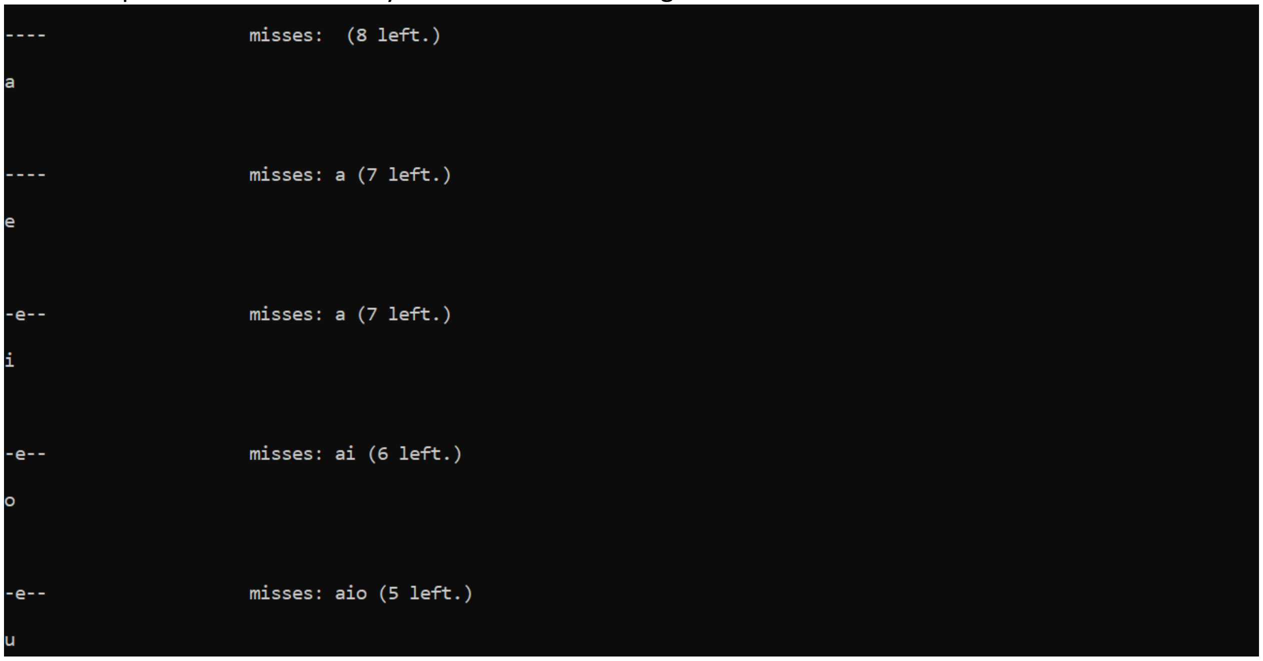
```
Please set the number of allowed misses:
w

Wrong input. Number of misses allowed starts from 1...
Please set the number of allowed misses:
8

Back to the menu. Choose below...
1. Choose the length of the word
2. Modify the difficulty level(Number of misses allowed)
0. Exit the game
```

The same happened with the first choice while choosing the length for a word. And, eventually, we succeeded to get a random word displayed on console window for us.



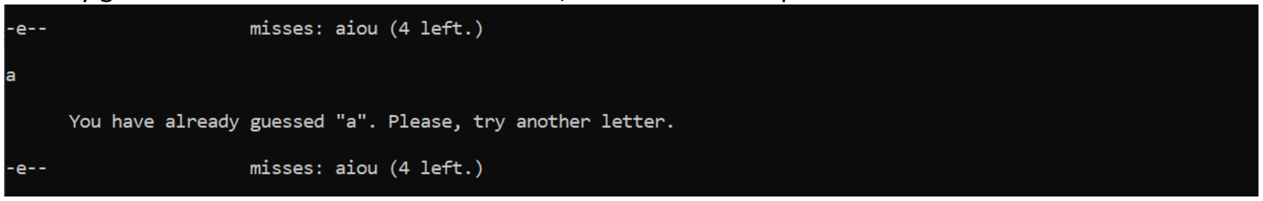
```
How many letters would you like a word to contain? (Range is from 1 to 16)
q


Out of range or incorrect input.
Please, choose the length of the word from the range

How many letters would you like a word to contain? (Range is from 1 to 16)
4



            Please, start guessing letter by letter
Bear in mind, the word might contain Upper or Lower case lettes



----                   misses:  (8 left.)

_
```

The main process went smoothly. We tried our best to guess it



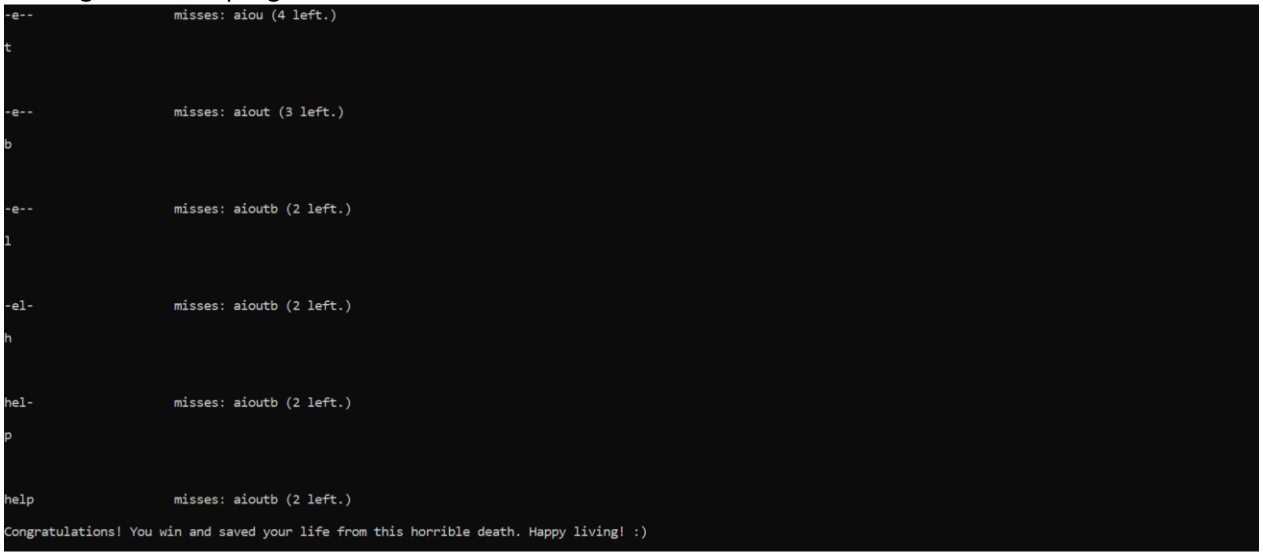
```
----                   misses:  (8 left.)

a



----                   misses: a (7 left.)

e



-e--                   misses: a (7 left.)

i



-e--                   misses: ai (6 left.)

o



-e--                   misses: aio (5 left.)

u
```

Misses were displayed in misses indicating how many more chances we have. On purpose was given an already guessed letter and for this time as well, the result is as expected



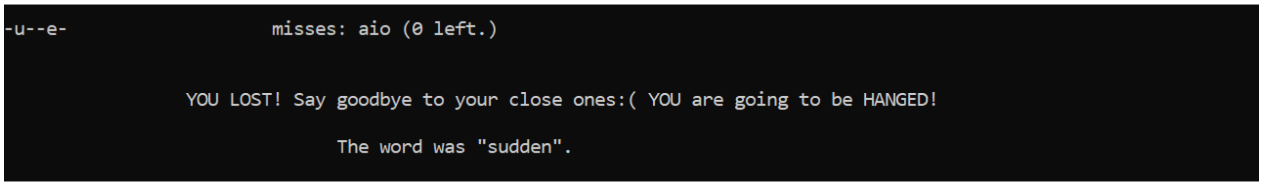
```
-e--                   misses: aiou (4 left.)

a

      You have already guessed "a". Please, try another letter.

-e--                   misses: aiou (4 left.)
```

After some more guesses we lucked out to guess the whole word correctly and get a “Congratulations” message from the program!



```
-e--                 misses: aiou (4 left.)

t


-e--                 misses: aiout (3 left.)

b


-e--                 misses: aioutb (2 left.)

l


-el-                 misses: aioutb (2 left.)

h


hel-                 misses: aioutb (2 left.)

p


help                 misses: aioutb (2 left.)

Congratulations! You win and saved your life from this horrible death. Happy living! :)
```

Next time, we failed to guess so the program gave a result of HANGING US!



```
-u--e-                 misses: aio (0 left.)


                YOU LOST! Say goodbye to your close ones:( YOU are going to be HANGED!

                              The word was "sudden".
```