DEVELOPER DOC

Description of the program: Hangman

The program is a guessing game.

Only the user can choose the level of the difficulty that depends on the number of letters in the word to guess and the allowed number of misses.

At first the program will print dashes on the screen and with every right guess the letter will be printed instead of the dash in its right position. If the letter occurs more than once in the word, then all dashes covering this letter will be unveiled. If the suggested letter does not occur in the word, it counts as a miss.

If the user enters a letter that he has already guessed, he will be notified and it will not be counted as a mistake.

The user loses if the number of the wrongly guessed letters surpassed the allowed number of misses.

In each try, the word must be selected randomly from the file that is attached to the problem and should satisfy the level of difficulty chosen by the user.

INT MAIN( ):

```
char* word;
char* dashes;
char letter;
int miss=0,allowed_miss=0,word_length,i,restart;
```

The declared variables in main are:
-word: the string that will contain the word to guess. It is dynamically allocated because the user will enter its length later.
-dashes: the string that will hide every letter of the word with a dash. It has the same length as word, that is why it is dynamically allocated.
-letter: will contain the character that the user will enter in each try.
-miss: an integer that will contain the number of the mistakes that the user will do.
-allowed_miss: an integer that contains the number of the allowed mistakes defined by the user.
-word_length: an integer that will contain the length of the world defined by the user.
-i: a counter that I will use later in this function.
-restart: it will contain 0 or 1: based on the decision of the user, if he wants to continue playing, he enters 1 otherwise he enters 0.

```
do
{   fflush(stdin);
    printf("Word's length : ");
    scanf("%d",&word_length);
    if ((word_length<2)||(word_length>15))
    printf("Error, re-enter the word's length, it should be a number between 2 and 15.\n");
}
while ((word_length<2)||(word_length>15));
```

In this part, I wanted to make sure that the user enters a number that is between 2 and 15.
(2 is the minimum length of the word while 15 is the maximum).
To do so, I used a do … while loop.
Inside the loop, the program will read the length: the program will keep reading the length until it is a number between 2 and 15
If the user entered a number, that is not between the two numbers, or a character he will be notified to re-enter the length.
Once, the user enters a character instead of a number the loop will never end, that is why I used fflush(stdin) so that the user will be able to re-enter the length once again.

```c
do
{    fflush(stdin);
     printf("Allowed number of misses : ");
     scanf("%d",&allowed_miss);
    if (allowed_miss<=0)
    printf("Error, re-enter the allowed number of misses, it should be a number higher than 0.\n");
}
while (allowed_miss<=0);
```

This loop Is the same as the previous one. The only difference is that the allowed number of misses should be a number higher than 0.

```c
word=(char*)malloc(word_length*sizeof(char));
word=random_word(word_length);
dashes=(char*)malloc(word_length*sizeof(char));

for(i=0; i<word_length; i++)
    dashes[i]='-';
dashes[i]='\0';
```

-Allocation of the two string: word and dashes (both have the same length).
-word: will contain the word selected randomly by the function random_word(word_length).
-dashes: will contain dashes that will hide the letters in the word.

```c
while((miss<allowed_miss) && (pos_letter(dashes,'-')>=0))
{
    printf("current word: ");
    puts(dashes);
    do
    {   fflush(stdin);
        printf("\nEnter the letter: ");
        scanf("%c",&letter);
        if ((toupper(letter)<'A')||(toupper(letter)>'Z'))
            printf("Error. enter a letter\n");
    }
    while ((toupper(letter)<'A')||(toupper(letter)>'Z'));
    printf("\n");
    if (pos_letter(word,letter)>=0)
    {
        if (pos_letter(dashes,letter)>=0)
        {
            printf("You have already guessed this letter\n");
        }
        else
        {
            printf("You guessed correctly !\n");
            unveil(word,dashes,letter);
        }
    }
    else
    {
        miss=miss+1;
        printf("Wrong letter ! You have %d tries left\n \n",allowed_miss-miss);
    }
}
```

This while loop will end once the mistakes done by the user (miss) are equal to the allowed number of mistakes (allowed_miss) or the dashes will not contain any dash anymore(meaning that he guessed the word correctly).
At the beginning the program will print dashes, then asks the user to enter a letter. Here, comes the importance of the do … while loop because it guarantees that the user enters only a character in each try.
If the input is not a character, the user will be notified to re-enter the letter and it will not be counted as a mistake since the input is not even a character. After the user enters the letter , the program will check if it occurs in the word with the pos_letter(word,letter) function.
If it does, the program will check if the letter was already guessed with pos_letter(word,dashes): if it does, the user will be notified and it will not be counted as a mistake but if it does not the letter will be considered as a right guess and the dashes covering this letter will be unveiled with the unveil(word,dashes,letter) function.
If the function does not exist in the word then miss will increase by 1.

```c
if (miss<allowed_miss)
    printf("Congratulations, you won.\n");
else printf("Unfortunately, you lost.\n");
printf("The word is: ");
puts(word);
free(word);
free(dashes);
printf("If you wish to play again press (1) if you want to quit press (0)\n");
scanf("%d",&restart);
if (restart==1)
    main();
return 0;
```

If the number of the mistakes(miss) is less than the allowed number of mistakes (allowed_miss), it means that the previous while loop ended because there is no dashes left in dashes which means that the user guessed the word : then the user will be notified that he won. However, if it is equal, it means that the user did not guess the word, so he will be notified that he lost.
At the end, the memory of the word and the dashes will be released and the user will be asked if he wants to continue playing or not. If he wants to continue, he will press 1 which means that the main will be executed another time otherwise he will press 0 and the program ends.

CHAR* random_word(int word_length)

/*-This function guarantees the randomness of the word to guess.*/

```c
char* random_word(int word_length)
{
    int i=0,n=0;
    char* s;
    char* ch;
    s=(char*)malloc(word_length*sizeof(char));
    ch=(char*)malloc(word_length*sizeof(char));
    srand(time(NULL));
    n= rand()%200;
```

-The parameter of the function is the word length because we will need it to select the desired word.
-i : it is a counter.
-n: an integer that will contain a random number that is generated using srand() and rand()%200: rand () enables us to generate a random number but not in every execution. So, to do so I used srand() to make sure I get a random number every execution and I added %200 to make sure that the generated number is less than 200.
srand() needs a different variable as parameter in each execution to generate another random number and the only variable that changes in each execution is the time.
-s: will contain the string that will be read from the file each time.
-ch: will contain the string of the final word.

```
FILE* fp=fopen("WORDS.txt","r");
if (fp==NULL)
     return 0 ;
while(1)
{
     fscanf(fp,"%s",s);
     if (strlen(s) == word_length)
     {
          strcpy(ch,s);
           i=i+1;
     }
     if(i==n)
          break;
     if (fscanf(fp,"%s",s)!=1)
          break;
}
fclose(fp);
return ch;
```

After opening the file the program will start reading each word in the file that will be stored in s: if the world's length is equal to the length desired by the user the word will be stored in ch and the integer i will be increased by 1.
When i becomes equal to n(the random number) the while loop will end which guarantees the randomness of the word.
If the random word generated by the program passes the number of the word with the same length then the function will return the last word with this length.

int pos_letter(char* word,char letter)

/*This function returns the position of the letter in the word whether it is a lowercase or uppercase letter. If the letter does not exist it returns -1*/

```
int pos_letter(char* word,char letter)
{
     int i;
     for(i=0; i<strlen(word); i++)
     {
          if (toupper(word[i])==toupper(letter))
               return i;
     }
     return -1;
}
```

The purpose of this function is to verify if a letter occurs in a word, that is why it has a word and a letter as parameters.
-i: it is a counter.
-during the for loop the program will go through the word and verify if there is a letter in the word that is similar to the one entered by the user.
if the letter occurs in the word then the function will return its position otherwise the function will return -1.

void unveil(char* word,char* dashes,char letter)

/*This function replaces every dash with correctly guessed letter*/

```
void unveil(char* word,char* dashes,char letter)
{
     int i;
     for(i=0; i<strlen(dashes); i++)
     {
          if (toupper(word[i])==toupper(letter))
          {
               dashes[i]=word[i];
          }
     }
}
```

-This function will reveal the dashes covering the letters in their right position and to do so we need the dashes, the word and the letter as parameters.
i: it is a counter.
During the for loop the program will go through the word and in each time the letter in the word is similar to the entered letter the dash in the same position will be replaced by this letter.
I used toupper() so that the user does not worry about whether the letter is uppercase or undercase.