Functions Pointers Arrays
Seneca IPC Inventory System Main menu

Workshop 6 (in lab) Final Project Milestone 1 (at home)

In this workshop, you are to code several input/output functions to perform fool-proof entry from keyboard. These functions later will be used to create the main menu of the Final project.

IN-LAB :

Implement the following function in tools.c file:

```
void flushKeys(void);
```
Flushes the keyboard by reading characters one by one until it reads the newline character.

```
void pause(void);
```
First is will print: `"Hit <ENTER> to continue...\n"`
Then it will call the **`flushKeys`** function.

```
int yes(void);
```
Returns true (1) if user enters y or Y and returns (0) if user enters n or N. Any other entry is rejected and flushed with the following error message:
`"Only (y) or (n) are acceptable, redo: "`

```
void scanInt(int* ip);
```
Performs a fool-proof integer entry. If the user enters anything other than a valid integer number it will print the following error message:
`"Invalid integer try again: "`
and try again until the user enters a valid value. Then it will set the target of the integer pointer argument to the valid integer value.

```
int getInt(int min, int max,
           const char* prompt, const char errMes[]);
```
Performs a fool-proof integer entry within the range given by min and max arguments. Getint first prints the prompt argument and then uses the scanInt function to read a valid integer.
If the value entered is outside the range given, it shows the following error message and the acceptable range:
`"%s (%d<= value <=%d): "`
and tries reading the integer again until the user enters a valid value. Then it will return the value.

```
void scanDouble(double* ip);
```
Works exactly like **scanInt** but for a double value, using the following error message:
`"Invalid number try again: "`

```
double getDouble(double min, double max,
    const char* prompt, const char errMes[], int precision);
```

Works exactly like **getInt** but for double values. Note that **getDouble** also receives the printing precision of the double range values (`min` and `max`). The format string used for the error message is:
`"%s (%.*lf<= value <=%.*lf): "`

Use the tester program: 144_ms1_lab_tester.c to test your functions. The output of the tester should be exactly as follows: (Red values are what user enters)

OUTPUT:

```
IPC144 ms1 in lab test
Starting tester!
Hit <ENTER> to continue...

Testing yes(), Enter the followig:
abc<ENTER>
y<ENTER>
abc
Only (y) or (n) are acceptable, redo: y
```

```
Passed!
Enter: Y<ENTER>
Y
Passed!
Enter: n<ENTER>
n
Passed!
Enter: N<ENTER>
N
Passed!
Testing scanInt(), Enter the following:
abc<ENTER>
1abc<ENTER>
1<ENTER>

abc
Invalid integer try again: 1abc
Invalid integer try again: 1
Entered value is: 1

The output should be exactly as follows:

abc
Invalid integer try again: 1abc
Invalid integer try again: 1
Entered value is : 1

End scanInt() test

Tesing getint(), Enter the following:
a<ENTER>
1<ENTER>
3<ENTER>
2<ENTER>

> a
Invalid integer try again: 1
Bad 2 (2<= value <=2): 3
Bad 2 (2<= value <=2): 2
Entered value is: 2

The output should be exactly as follows:

> a
Invalid integer try again: 1
Bad 2 (2 <= value <= 2) : 3
Bad 2 (2 <= value <= 2) : 2
```

```
Entered value is : 2

End getInt() test

Testing scanDouble(), Enter the following:
abc<ENTER>
1abc<ENTER>
1<ENTER>

abc
Invalid number try again: 1abc
Invalid number try again: 1
Entered value is: 1.0

The output should be exactly as follows:

abc
Invalid number try again: 1abc
Invalid number try again: 1
Entered value is : 1.0

End scanDouble() test

Tesing getDouble(), Enter the following:
a<ENTER>
1<ENTER>
6<ENTER>
4<ENTER>

> a
Invalid number try again: 1
Bad 4 (3.0<= value <=5.0): 6
Bad 4 (3.0<= value <=5.0): 4
Entered value is: 4.0

The output should be exactly as follows:

> a
Invalid number try again: 1
Bad 4 (3.0 <= value <= 5.0) : 6
Bad 4 (3.0 <= value <= 3.0) : 4
Entered value is : 4.0

End getDouble() test

End IPC144 ms1 in lab test
```

IN-LAB SUBMISSION

To test and demonstrate execution of your program use the same data as the output example above.

Upload `tools.c` and `tools.h` and `144_ms1_lab_tester.c` to your matrix account. Compile and run your code and make sure everything works properly.

Then, run the following script from your account: (use your professor's Seneca `userid` to replace `profname.proflastname`)

`~fardad.soleimanloo/submit 144_ms1_lab``<ENTER>`

and follow the instructions.

> **IMPORTANT**: Please note that a successful submission does not guarantee full credit for this workshop. If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Resubmissions will attract a penalty.

AT-HOME (30%)

```
Under consturction.
```

QUIZ REFLECTION

Add a section to `reflect.txt` called "**Quiz X Reflection:**" (replace the '**X**' with the number of the last quiz).

Identify all of the questions from the quiz that you did not answer correctly. Under each question, specify the correct answer. If you missed the last quiz, enter all of the questions and the correct answer for each.

AT-HOME SUBMISSION

To test and demonstrate execution of your program, use the same data as the output example above.

If not on matrix already, upload `reflect.txt`, `menu.c, menu.h, tools.c, tools.h` and `inventory.c` to your matrix account. Compile and run your code and make sure everything works properly.

Then, run the following command from your account:

`~fardad.soleimanloo/submit 144_ms1_home``<ENTER>`

and follow the instructions.

IMPORTANT: Please note that a successful submission does not guarantee full credit for this workshop. If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Resubmissions will attract a penalty.