Programming Project 1 – Input Validation

Created by

Mr.Lutfee Deemae (Lut)

Student ID: 63070503448

Computer Engineering (International Program)

Task

Task 1: Choice 4 - Validate date in form dd MMM yyyy Western year

Task 2: Choice 12 - Validate CPE student ID number

Task 3: Choice 16 - Validate a string as a C language identifier

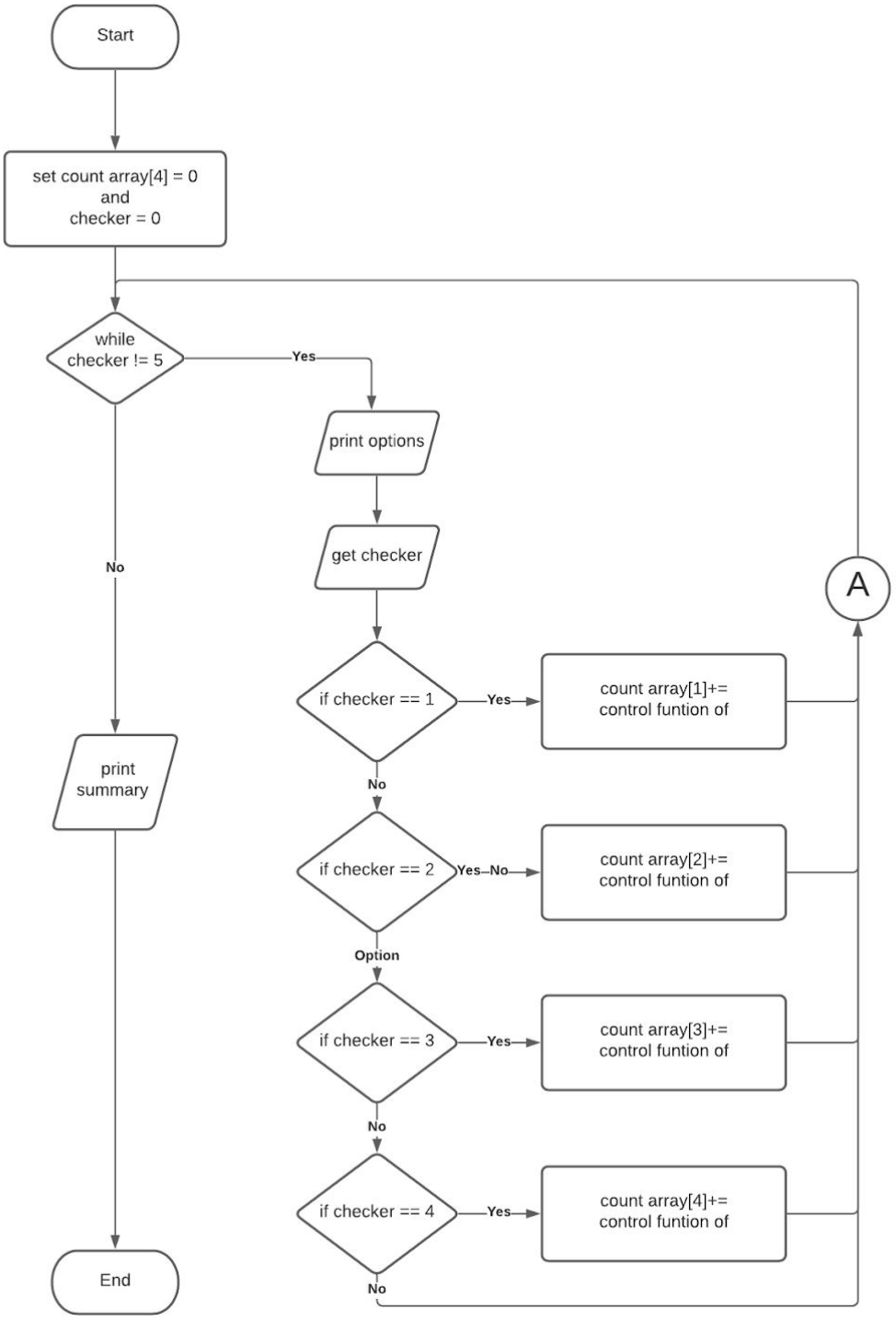
Task 4: Choice 7 - Validate international phone number

CPE100 International Sections, 1/2020

King Mongkut's University of Technology Thonburi

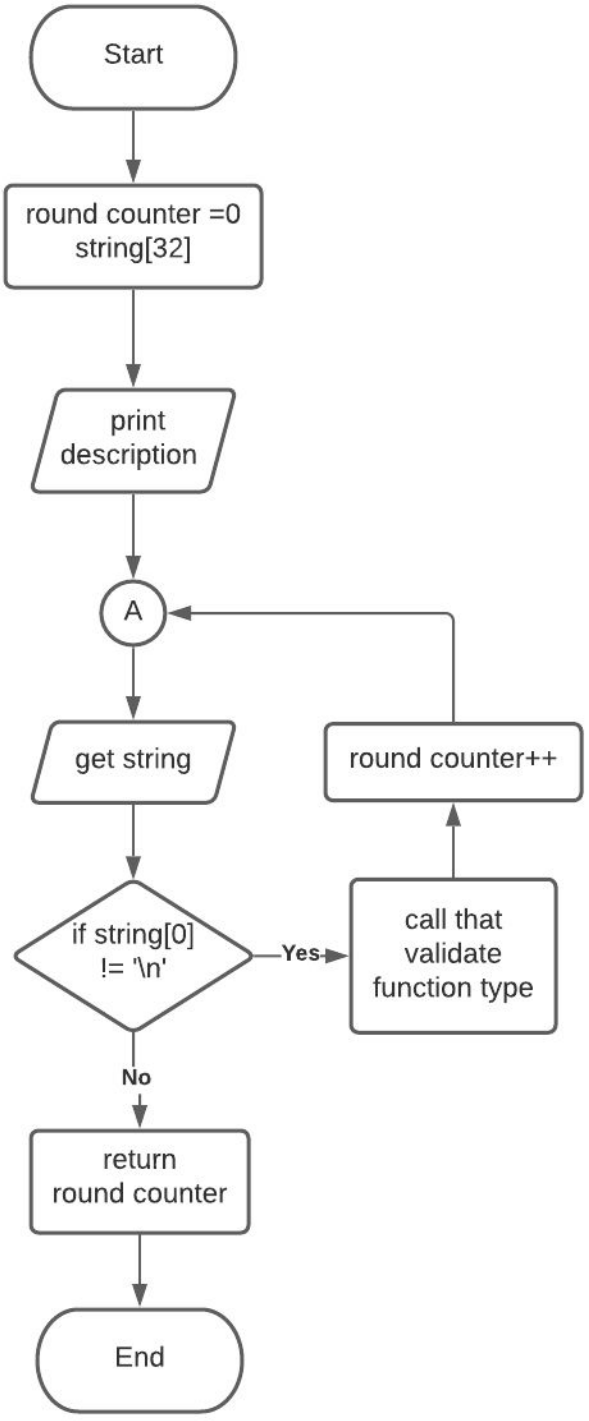
Main Function

Flowchart



Control Function

Flowchart



Task 1: Choice 4

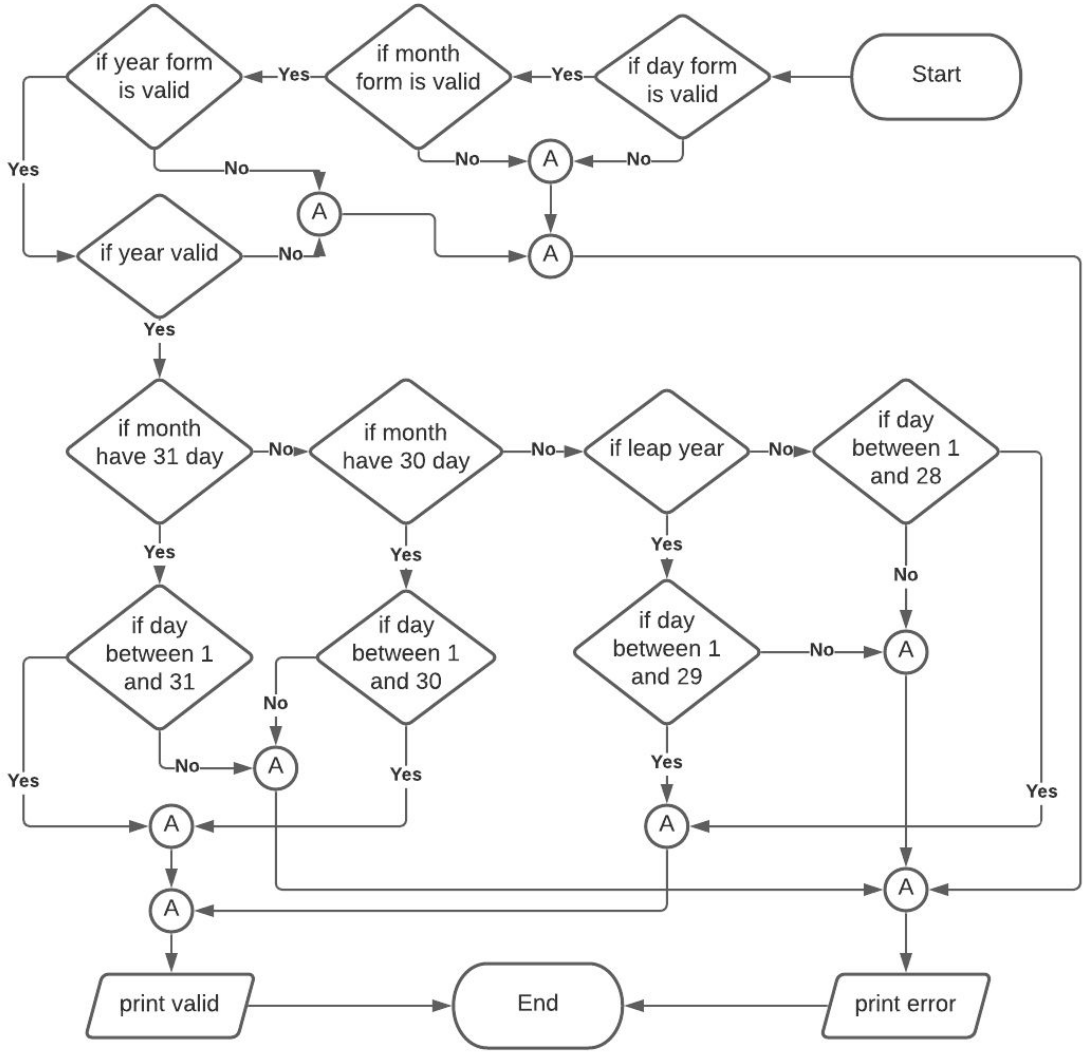
Validate date in form dd MMM yyyy Western year

Proposition:

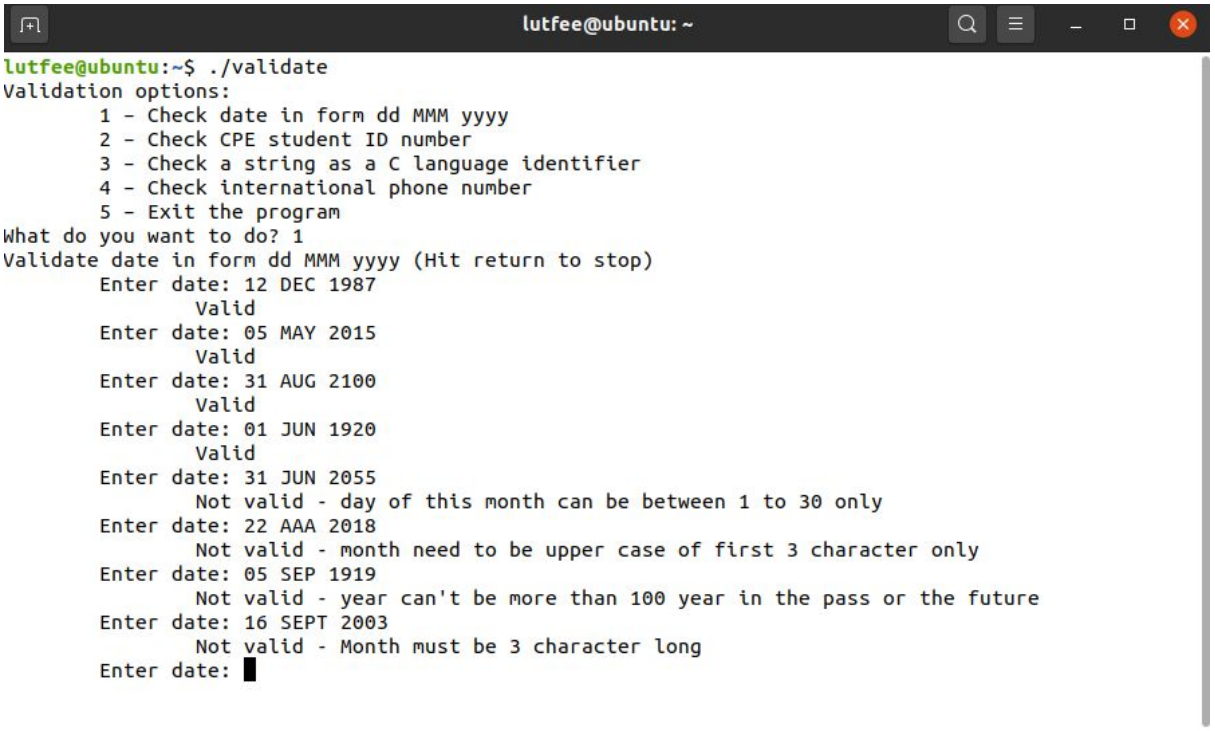
'MMM' is first 3 chars of month, capital letters. As above, you must make sure that the date is both formatted correctly and a legal date. Years cannot be more than 100 years in the past or the future.

Valid Input	Invalid Input
12 DEC 1987	31 JUN 2055 (invalid day of month)
05 MAY 2015	22 AAA 2018 (invalid month)
31 AUG 2100	05 SEP 1919 (year outside allowed range)
01 JUN 1920	16 SEPT 2003 (invalid month abbreviation)

Flowchart


Start
if day form is valid
if month form is valid
if year form is valid
if year valid
if month have 31 day
if month have 30 day
if leap year
if day between 1 and 28
if day between 1 and 31
if day between 1 and 30
if day between 1 and 29
A
Yes
No
print valid
End
print error

Result



Test Result

Validate date in form dd MMM yyyy (Hit return to stop)
Enter date: 12 DEC 1987
Valid
Enter date: 05 MAY 2015
Valid
Enter date: 31 AUG 2100
Valid
Enter date: 01 JUN 1920
Valid
Enter date: 31 JUN 2055
Not valid - day of this month can be between 1 to 30 only
Enter date: 22 AAA 2018
Not valid - month need to be upper case of first 3 character only
Enter date: 05 SEP 1919
Not valid - year can't be more than 100 year in the pass or the future
Enter date: 16 SEPT 2003
Not valid - Month must be 3 character long
Enter date:

Task 2: Choice 12

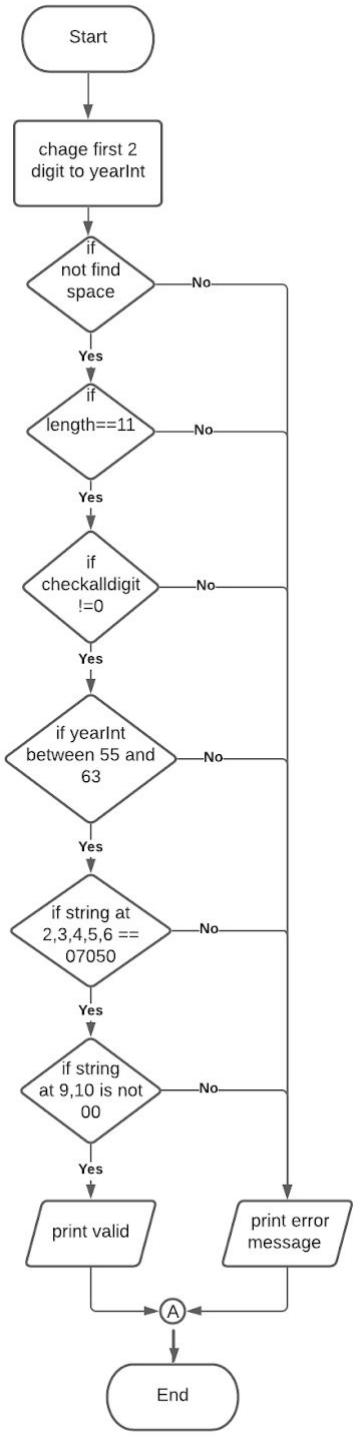
Validate CPE student ID number

Proposition:

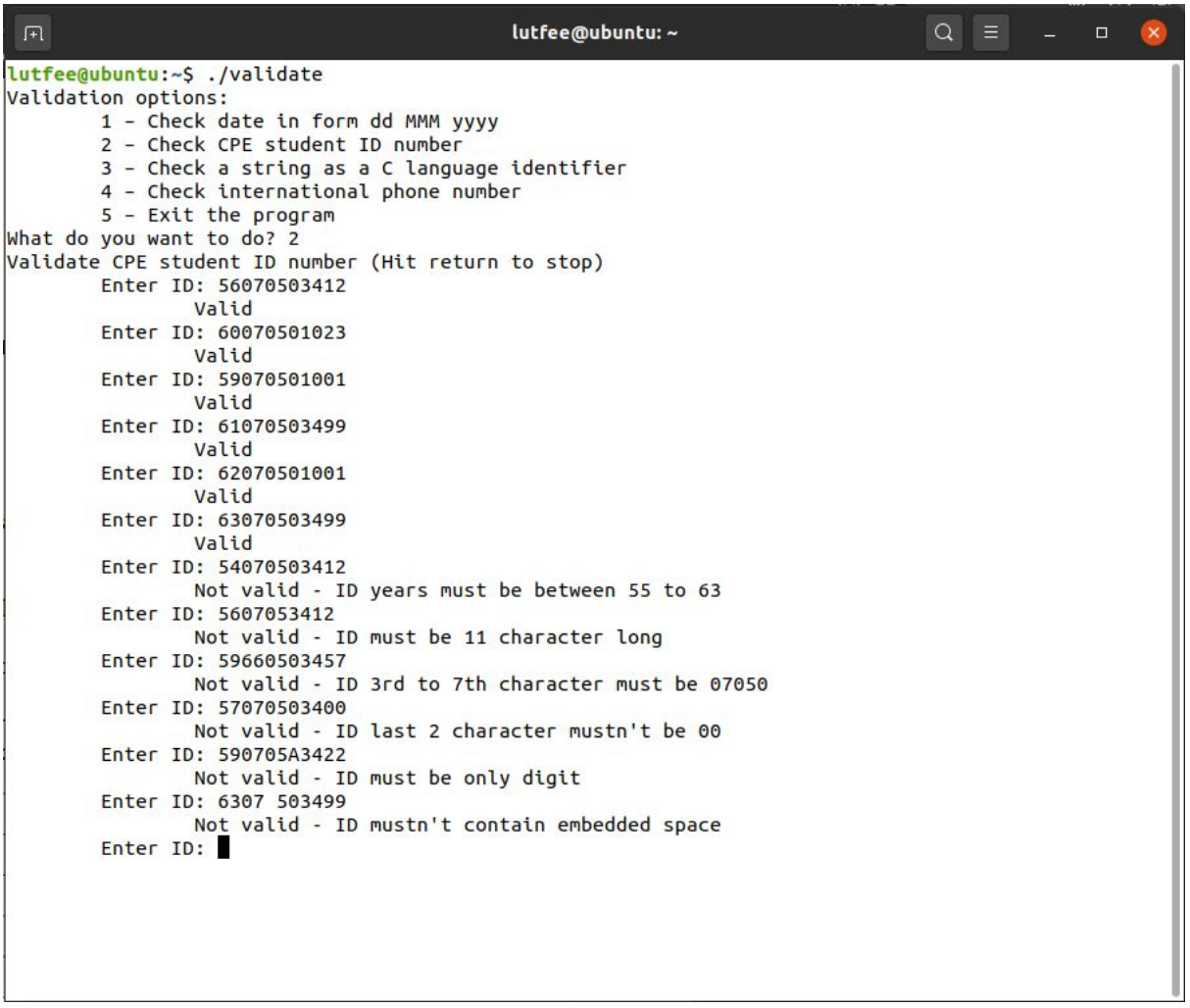
The ID numbers used in CPE for the past eight years are 11 digits long. They have the following form: yy07050ppdd. The ‘yy’ is the last two digits of the B.E. year in which the student entered KMUTT, for example, 55 would mean the student started in 2555. The ‘pp’ depends on whether the student is an International (34) or Thai program (10) student. The ‘dd’ can be any two digits except ‘00’. Your validation function should accept as valid any student ID number for years from 2555 to 2563 that follows this pattern. Any other pattern should be an error.

Valid Input	Invalid Input
56070503412	54070503412 (invalid year)
60070501023	5607053412 (too short)
59070501001	59660503457 (wrong middle digits)
61070503499	57070503400 (ends in 0)
62070501001	590705A3422 (contains alphabetic character)
63070503499	6307 503499 (embedded space)

Flowchart


Start
chage first 2 digit to yearInt
if not find space
No
Yes
if length==11
No
Yes
if checkalldigit !=0
No
Yes
if yearInt between 55 and 63
No
Yes
if string at 2,3,4,5,6 == 07050
No
Yes
if string at 9,10 is not 00
No
Yes
print valid
print error message
A
End

Result


lutfee@ubuntu: ~
lutfee@ubuntu:~$ ./validate
Validation options:
1 - Check date in form dd MMM yyyy
2 - Check CPE student ID number
3 - Check a string as a C language identifier
4 - Check international phone number
5 - Exit the program
What do you want to do? 2
Validate CPE student ID number (Hit return to stop)
Enter ID: 56070503412
Valid
Enter ID: 60070501023
Valid
Enter ID: 59070501001
Valid
Enter ID: 61070503499
Valid
Enter ID: 62070501001
Valid
Enter ID: 63070503499
Valid
Enter ID: 54070503412
Not valid - ID years must be between 55 to 63
Enter ID: 5607053412
Not valid - ID must be 11 character long
Enter ID: 59660503457
Not valid - ID 3rd to 7th character must be 07050
Enter ID: 57070503400
Not valid - ID last 2 character mustn't be 00
Enter ID: 590705A3422
Not valid - ID must be only digit
Enter ID: 6307 503499
Not valid - ID mustn't contain embedded space
Enter ID:

Test Result

What do you want to do? 2

Validate CPE student ID number (Hit return to stop)

Enter ID: 56070503412

Valid

Enter ID: 60070501023

Valid

Enter ID: 59070501001

Valid

Enter ID: 61070503499

Valid

Enter ID: 62070501001

Valid

Enter ID: 63070503499

Valid

Enter ID: 54070503412

Not valid - ID years must be between 55 to 63

Enter ID: 5607053412

Not valid - ID must be 11 character long

Enter ID: 59660503457

Not valid - ID 3rd to 7th character must be 07050

Enter ID: 57070503400

Not valid - ID last 2 character mustn't be 00

Enter ID: 590705A3422

Not valid - ID must be only digit

Enter ID: 6307 503499

Not valid - ID mustn't contain embedded space

Enter ID:

Task 3: Choice 16

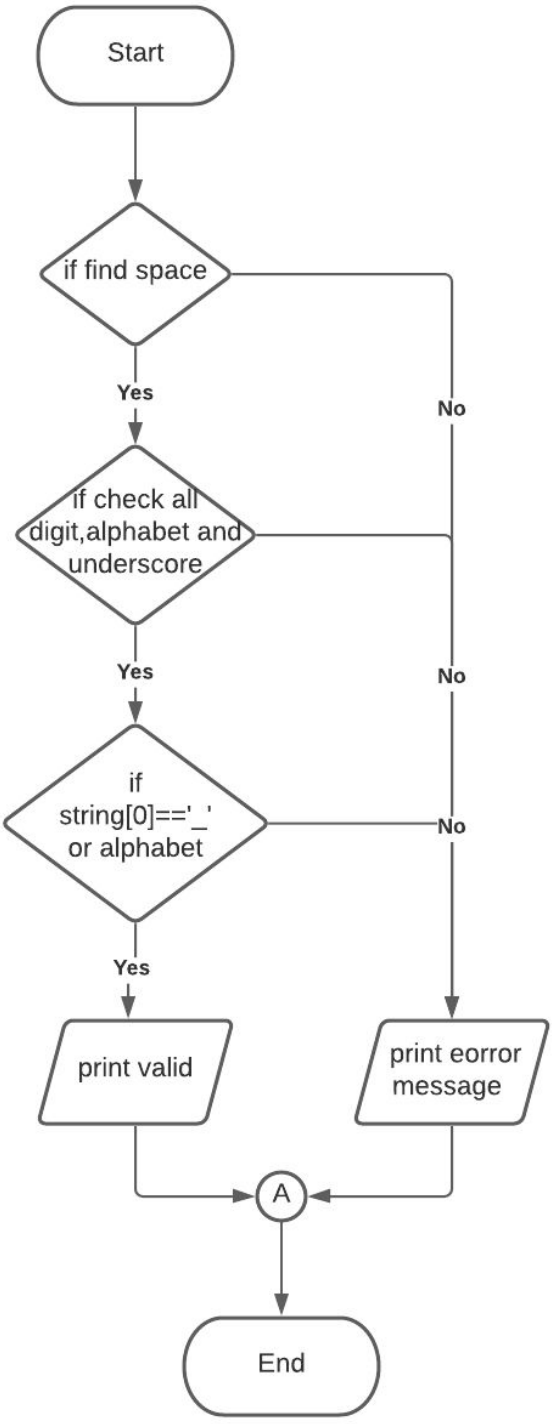
Validate a string as a C language identifier

Proposition:

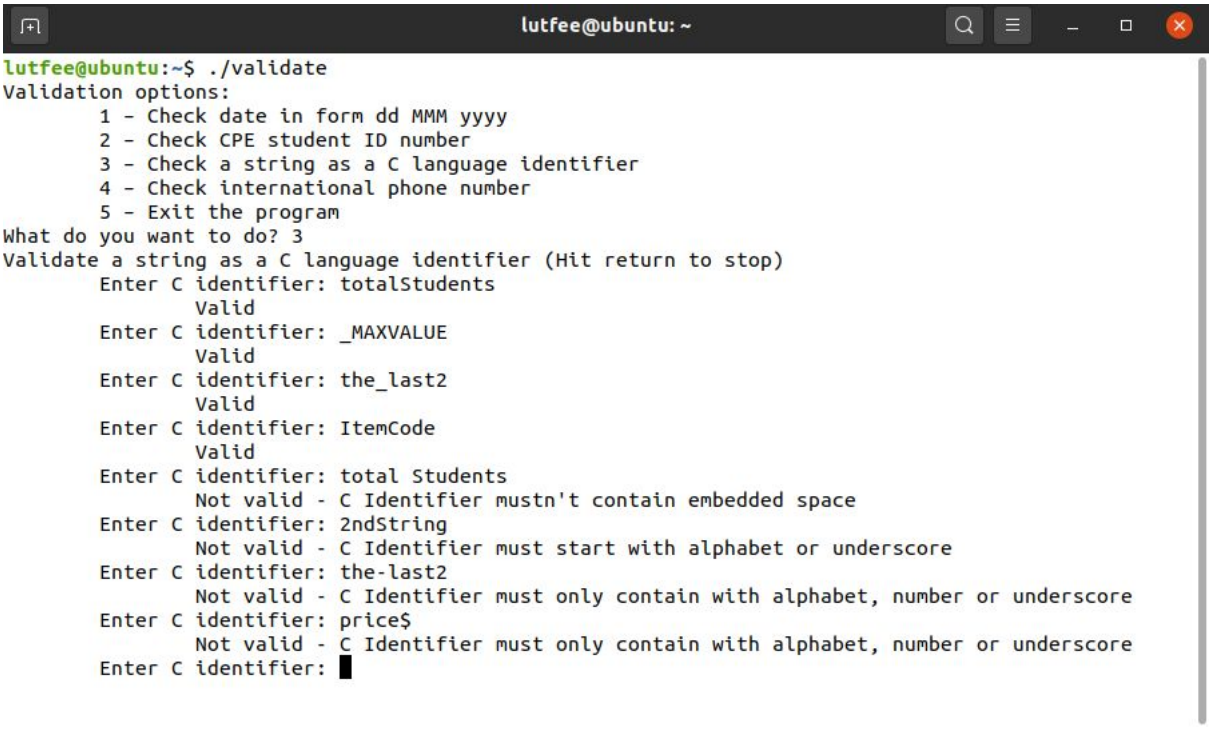
In Lecture 2 we studied the rules for valid C identifiers. Your function should correctly distinguish legal from illegal C identifiers. (This validation should NOT consider our coding standard rules.)

Valid Input	Invalid Input
totalStudents	total Students (embedded space)
__MAXVALUE	2ndString (starts with number)
the_last2	thelast2 (dash is not legal)
ItemCode	price$ (dollar sign is not legal)

Flowchart


Start
if find space
Yes
No
if check all digit,alphabet and underscore
Yes
No
if string[0]=='_' or alphabet
No
Yes
print valid
print eorror message
A
End

Result


lutfee@ubuntu: ~
lutfee@ubuntu:~$ ./validate
Validation options:
1 - Check date in form dd MMM yyyy
2 - Check CPE student ID number
3 - Check a string as a C language identifier
4 - Check international phone number
5 - Exit the program
What do you want to do? 3
Validate a string as a C language identifier (Hit return to stop)
Enter C identifier: totalStudents
Valid
Enter C identifier: _MAXVALUE
Valid
Enter C identifier: the_last2
Valid
Enter C identifier: ItemCode
Valid
Enter C identifier: total Students
Not valid - C Identifier mustn't contain embedded space
Enter C identifier: 2ndString
Not valid - C Identifier must start with alphabet or underscore
Enter C identifier: the-last2
Not valid - C Identifier must only contain with alphabet, number or underscore
Enter C identifier: price$
Not valid - C Identifier must only contain with alphabet, number or underscore
Enter C identifier:

Test Result

Validate a string as a C language identifier (Hit return to stop)

Enter C identifier: totalStudents

Valid

Enter C identifier: _MAXVALUE

Valid

Enter C identifier: the_last2

Valid

Enter C identifier: ItemCode

Valid

Enter C identifier: total Students

Not valid - C Identifier mustn't contain embedded space

Enter C identifier: 2ndString

Not valid - C Identifier must start with alphabet or underscore

Enter C identifier: the-last2

Not valid - C Identifier must only contain with alphabet, number or underscore

Enter C identifier: price$

Not valid - C Identifier must only contain with alphabet, number or underscore

Enter C identifier:

Task 4: Choice 7

Validate international phone number

Proposition:

A valid international phone number must follow these rules:

• Only digits, dashes and one plus sign (+) allowed (no spaces)

• Must begin with a plus

• After the plus, must have between 1 and 3 digits for the country code

• After the country code, must have a dash

• After the dash, must have between 1 and 3 digits for a city/area code

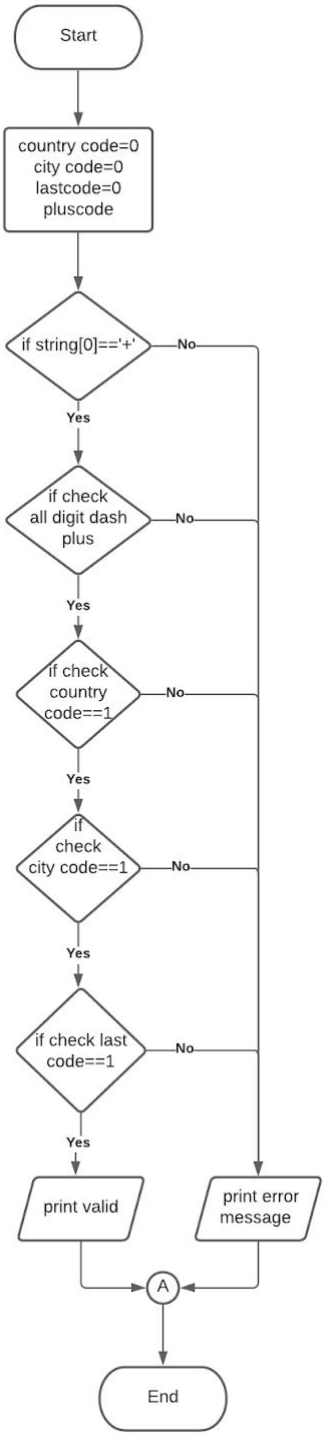
• After the area code, must have a dash

• After the second dash, must be between 6 and 10 additional digits.

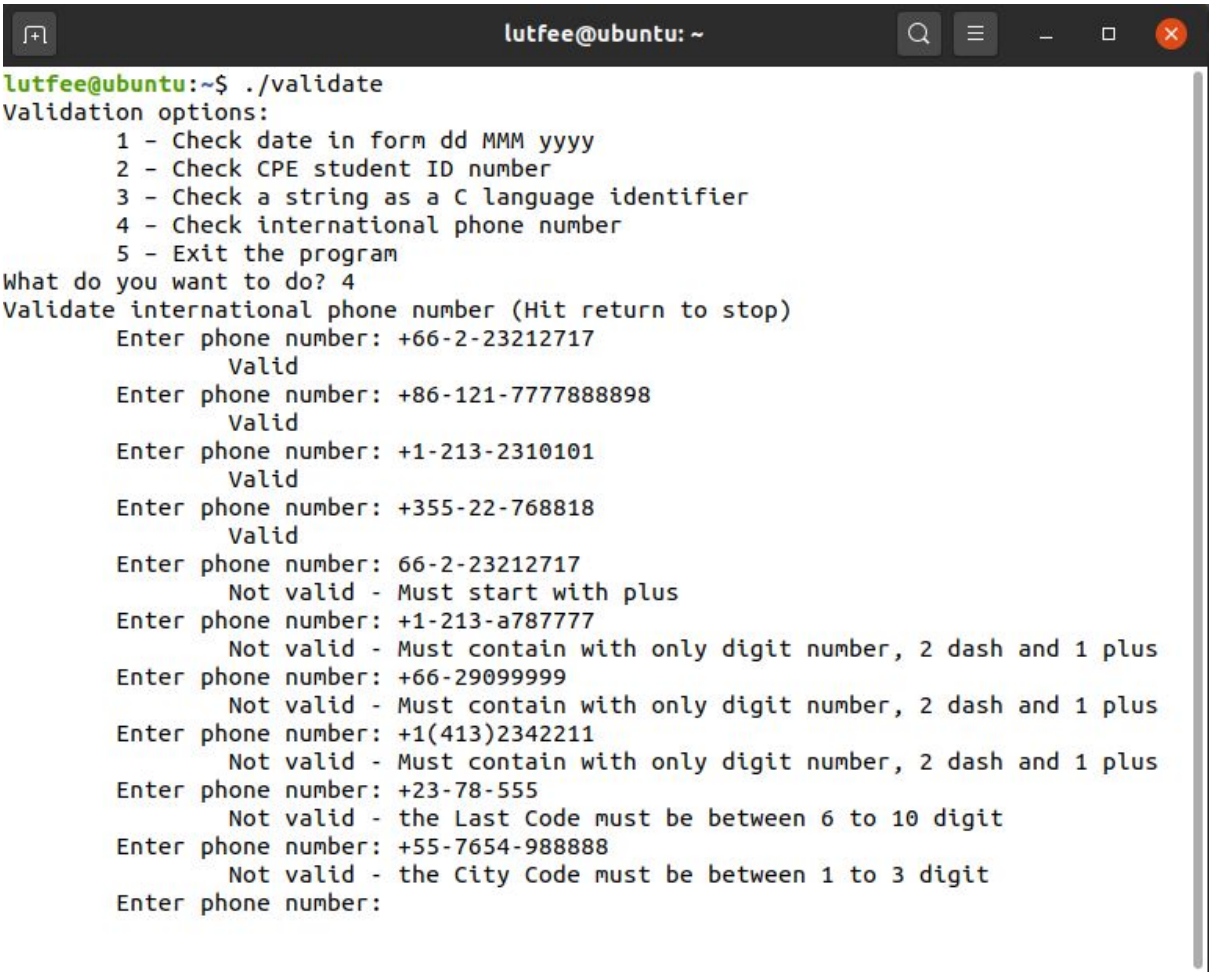
You do not need to check that the country code is the code for a real country.

Valid Input	Invalid Input
+66223212717	66223212717 (no plus)
+861217777888898	+1213a787777 (alphabetic character)
+12132310101	+6629099999 (missing dash)
+35522768818	+1(413)2342211 (parentheses not allowed)
	+2378555 (too short)
	+557654988888 (city/area code too long)

Flowchart


Start
country code=0
city code=0
lastcode=0
pluscode
if string[0]=='+'
No
Yes
if check
all digit dash
plus
No
Yes
if check
country
code==1
No
Yes
if
check
city code==1
No
Yes
if check last
code==1
No
Yes
print valid
print error
message
A
End

Result


lutfee@ubuntu: ~
lutfee@ubuntu:~$ ./validate
Validation options:
1 - Check date in form dd MMM yyyy
2 - Check CPE student ID number
3 - Check a string as a C language identifier
4 - Check international phone number
5 - Exit the program
What do you want to do? 4
Validate international phone number (Hit return to stop)
Enter phone number: +66-2-23212717
Valid
Enter phone number: +86-121-7777888898
Valid
Enter phone number: +1-213-2310101
Valid
Enter phone number: +355-22-768818
Valid
Enter phone number: 66-2-23212717
Not valid - Must start with plus
Enter phone number: +1-213-a787777
Not valid - Must contain with only digit number, 2 dash and 1 plus
Enter phone number: +66-29099999
Not valid - Must contain with only digit number, 2 dash and 1 plus
Enter phone number: +1(413)2342211
Not valid - Must contain with only digit number, 2 dash and 1 plus
Enter phone number: +23-78-555
Not valid - the Last Code must be between 6 to 10 digit
Enter phone number: +55-7654-988888
Not valid - the City Code must be between 1 to 3 digit
Enter phone number:

Test Result

Validate international phone number (Hit return to stop)

Enter phone number: +66-2-23212717

Valid

Enter phone number: +86-121-7777888898

Valid

Enter phone number: +1-213-2310101

Valid

Enter phone number: +355-22-768818

Valid

Enter phone number: 66-2-23212717

Not valid - Must start with plus

Enter phone number: +1-213-a787777

Not valid - Must contain with only digit number, 2 dash and 1 plus

Enter phone number: +66-29099999

Not valid - Must contain with only digit number, 2 dash and 1 plus

Enter phone number: +1(413)2342211

Not valid - Must contain with only digit number, 2 dash and 1 plus

Enter phone number: +23-78-555

Not valid - the Last Code must be between 6 to 10 digit

Enter phone number: +55-7654-988888

Not valid - the City Code must be between 1 to 3 digit

Enter phone number:

Source Code

```
/*****************************************************************
 *
 *      validate
 *
 *              This program is   will ask user what type of information they
will validate and then it will
 *              ask the user for the information that will be validated and
then return the validation result
 *              back to the user interface
 *
 *              Created by Lutfee Deemae (Lut) ID 63070503448
 *              29 September 2020
 *
 *****************************************************************
 */

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>

#define NEWLINE 10
#define BACKSPACE 32
/*
        This function is a function for printing user interface GUI  choices
*/
void printOption();

/*
        This function is a function for printing summary result of each kind
of task that had execute
        - 1 argument which is array of summation in each kind of task
*/
void printSummary(int catagoryValidateCount[]);

/*
```

```
	This function is a control function for a date validation that will loop
until user enter onliy newline
	- return a variable that count round of execution
*/
int checkDates();

/*
	This function is a control function for a id validation that will loop
until user enter onliy newline
	- return a variable that count round of execution
*/
int checkIDs();

/*
	This function is a control function for a c identifier validation that
will loop until user enter onliy newline
	- return a variable that count round of execution
*/
int checkIdentifiers();

/*
	This function is a control function for a phone validation that will
loop until user enter onliy newline
	- return a variable that count round of execution
*/
int checkPhones();

/*
	This function is a validation function of date in forn of dd/MMM/yyy
	that will check that string is valid or not
	- 1 argument which is string that will be check
*/
void validDates(char stringToValidate[]);

/*
	This function is a validation function of id that will check that string
is valid or not
```

```
        - 1 argument which is string that will be check
*/
void validIDs(char stringToValidate[]);

/*
        This function is a validation function of c identifier that will check
that string is valid or not
        - 1 argument which is string that will be check
*/
void validIdentifiers(char stringToValidate[]);

/*
        This function is a validation function of international phone that will
check that string is valid or not
        - 1 argument which is string that will be check
*/
void validPhones(char stringToValidate[]);

/*
        This function will check that it's can find space in that string or not
        - return 1 if find a space, 0 if not find any space in string.
        - 1 argument which is string that will be check
*/
int findSpace(char stringToFind[]);

/*
        This function will check that it's can find only digit in that function or
not
        - return 1 if all character in string is digit, 0 if not
        - 1 argument which is string that will be checks
*/
int checkAllDigits(char stringToCheck[]);

/*
        This function will check that it's find only digit, alphabet or
underscore in that string or not
        - return 1 if yes, 0 if not
```

```
        - 1 argument which is string that will be check
*/
int checkAllNumberAlphabetUnderscore(char stringToCheck[]);

/*
        This function will check that it's find only digit, dash or plus in that string or not
        and check that in that string contain 1 of plus sign and 2 of dashes sign or not
        - return 1 if yes, 0 if not
        - 1 argument which is string that will be check
*/
int checkAllNumberDashPlus(char stringToCheck[]);

/*
        This function will check string from the start position to atleast min size of that expected and
        maximum at that expected toget and check that section is end by expected character or not
        - return the size of that section of string if that section is valid.
                if not, return -1 if it's contain others thing that's not digit or it's size less than minimum expected
                or, return -2 if it size is greater than maximum expected size
        - 5 argument:
                1-string that will be check
                2-start position that will check of that string
                3-minimum size that expected that section to be
                4-maximum size that expected that section to be
                5-expected character to be at the end of that section
*/
int checkPhoneNumberCodeBySection(char stringToCheck[],int startPosition,int minSizeOfSection,int maxSizeOfSection,char expectedEndingCharacter);

/*
        This function will check that it's find only digit, uppercase alphabet or space in that string or not
```

```
        - return 1 if yes, 0 if not
        - 1 argument which is string that will be check
*/
int checkAllNumberUpperAlphabetSpace(char stringToCheck[]);

/*
        This function will check day section in date string and check it's valid or not
        - return 0 if yes, return -1 if it's find uppercase character in the day section
                return -2 if it's find space in day section or find digit after day section instead of space
                return -3 if it's find uppercase character after the day section instead space
        - 1 argument which is date string that will be check
*/
int checkDaySection(char stringToCheck[]);

/*
        This function will check month section in date string and check it's valid or not
        - return 0 if yes, return -1 if it's find digit in the month section
                return -2 if it's find space in month section or find uppercase character after day section instead of space
                return -3 if it's find digit character after the day section instead space
        - 1 argument which is date string that will be check
*/
int checkMonthSection(char stringToCheck[]);

/*
        This function will check year section in date string and check it's valid or not
        - return 0 if yes, return -1 if it's find uppercase character in the day section
                return -2 if it's find space in day section or find that it's too short or too long for year
```

```
	- 1 argument which is date string that will be check
*/
int checkYearSection(char stringToCheck[]);

/*
	This function will check that date string is in the correct format or
not
	- return 1 if it's correct format
		return -1 to -9 if it's not valid
	- 1 argument which is date string that will be check
*/
int dateInputFormatValidation(char stringToValidate[]);

/*
	This function will check that date is correct or not
	- return 1 if it's correct format
		return -1 to -5 if it's not valid
	- 1 argument which is date string that will be check
*/
int dateValidationCheck(char stringToValidate[]);



int main(int argc,char* argv[])
{
	char terminalInput[32]; /* a variable to get a input from terminal that
user input */
	int catagoryValidateCount[4]={0,0,0,0};	/*  */
	int validateChecker;	/*  */
	validateChecker=0;
	while(validateChecker != 5)
	{
		printOption();
		fgets(terminalInput,sizeof(terminalInput),stdin);
		sscanf(terminalInput,"%d",&validateChecker);
		if(validateChecker==1)
		{
```

```
                catagoryValidateCount[0] += checkDates();
            }
            else if(validateChecker==2)
            {
                catagoryValidateCount[1] += checkIDs();
            }
            else if(validateChecker==3)
            {
                catagoryValidateCount[2] += checkIdentifiers();
            }
            else if(validateChecker==4)
            {
                catagoryValidateCount[3] += checkPhones();
            }
        }

        printSummary(catagoryValidateCount);

}

void printOption()
{
        printf("Validation options:\n");
        printf("\t1 – Check date in form dd MMM yyyy\n");
        printf("\t2 – Check CPE student ID number\n");
        printf("\t3 – Check a string as a C language identifier\n");
        printf("\t4 – Check international phone number\n");
        printf("\t5 – Exit the program \n");
        printf("What do you want to do? ");
}

void printSummary(int catagoryValidateCount[])
{
        printf("Program run summary:\n");
        for (int i = 0; i < 4; ++i)
        {
```

```
                printf("Validation type %d:  %d\n",
i+1,catagoryValidateCount[i]);
        }
        printf("\nGoodbye!\n");
}

int checkDates()
{
        char terminalInput[32]; /* a variable to get a input from terminal
that user input */
        char stringToValidate[32];      /* a variable for store a string that will
use to validated */
        int roundCounter;         /* a variable that will count haw many times
this work had run */
        roundCounter=0;

        printf("Validate date in form dd MMM yyyy (Hit return to stop)\n");

        while(1)
        {
                printf("\tEnter date: ");
                fgets(terminalInput,sizeof(terminalInput),stdin);
                if(terminalInput[0]==NEWLINE)
                {
                        break;
                }
                strcpy(stringToValidate,terminalInput);
                stringToValidate[strlen(stringToValidate)-1] = 0;
                validDates(stringToValidate);
                ++roundCounter;
        }

        return roundCounter;
}

int checkIDs()
{
```

```
	char terminalInput[32]; /* a variable to get a input from terminal
that user input */
	char stringToValidate[32];	/* a variable for store a string that will
use to validated */
	int roundCounter;		/* a variable that will count haw many times
this work had run */
	roundCounter=0;

	printf("Validate CPE student ID number (Hit return to stop)\n");

	while(1)
	{
		printf("\tEnter ID: ");
		fgets(terminalInput,sizeof(terminalInput),stdin);
		if(terminalInput[0]==NEWLINE)
		{
			break;
		}
		strcpy(stringToValidate,terminalInput);
		stringToValidate[strlen(stringToValidate)-1] = 0;
		validIDs(stringToValidate);
		++roundCounter;
	}

	return roundCounter;
}

int checkIdentifiers()
{
	char terminalInput[32]; /* a variable to get a input from terminal
that user input */
	char stringToValidate[32];	/* a variable for store a string that will
use to validated */
	int roundCounter;		/* a variable that will count haw many times
this work had run */
	roundCounter=0;
```

```
	printf("Validate a string as a C language identifier (Hit return to
stop)\n");

	while(1)
	{
		printf("\tEnter C identifier: ");
		fgets(terminalInput,sizeof(terminalInput),stdin);
		if(terminalInput[0]==NEWLINE)
		{
			break;
		}
		strcpy(stringToValidate,terminalInput);
		stringToValidate[strlen(stringToValidate)-1] = 0;
		validIdentifiers(stringToValidate);
		++roundCounter;
	}

	return roundCounter;
}

int checkPhones()
{
	char terminalInput[32]; /* a variable to get a input from terminal
that user input */
	char stringToValidate[32];	/* a variable for store a string that will
use to validated */
	int roundCounter;		/* a variable that will count haw many times
this work had run */
	roundCounter=0;

	printf("Validate international phone number (Hit return to stop)\n");

	while(1)
	{
		printf("\tEnter phone number: ");
		fgets(terminalInput,sizeof(terminalInput),stdin);
		if(terminalInput[0]==NEWLINE)
```

```
		{
			break;
		}
		strcpy(stringToValidate,terminalInput);
		stringToValidate[strlen(stringToValidate)-1] = 0;
		validPhones(stringToValidate);
		++roundCounter;
	}

	return roundCounter;
}

void validDates(char stringToValidate[])
{
	int inputFormValidation;	/* a variable that will hold a input form validation status */
	int validationCheck;	/* a variable that will hold a date is correct date or not */

	inputFormValidation = dateInputFormatValidation(stringToValidate);

	printf("\t\t");
	switch(inputFormValidation)
	{
		case -1:
			printf("Not valid - Dates must contain only with digit number, upper case alphabet and space\n");
			break;
		case -2:
			printf("Not valid - Day must contain only with digit number\n");
			break;
		case -3:
			printf("Not valid - Day must be 2 digit long\n");
			break;
		case -4:
```

```
                printf("Not valid - After the day must be space as in
form of (dd MMM yyyy)\n");
                break;
            case -5:
                printf("Not valid - Month must contain only with upper
case alphabet\n");
                break;
            case -6:
                printf("Not valid - Month must be 3 character long\n");
                break;
            case -7:
                printf("Not valid - After the month must be space as in
form of (dd MMM yyyy)\n");
                break;
            case -8:
                printf("Not valid - Year must contain only with digit
number\n");
                break;
            case -9:
                printf("Not valid - Year must be 4 digit long\n");
                break;
        }

        if (inputFormValidation==1)
        {
            validationCheck = dateValidationCheck(stringToValidate);
            //printf("inputFormValidation is %d\tvalidationCheck is %d\n",
inputFormValidation,validationCheck);
            switch(validationCheck)
            {
                case -1:
                    printf("Not valid - year can't be more than 100
year in the pass or the future\n");
                    break;
                case -2:
                    printf("Not valid - month need to be upper case of
first 3 character only\n");
```

```
                        break;
                case -3:
                        printf("Not valid - day of this month can be
between 1 to 31 only\n");
                        break;
                case -4:
                        printf("Not valid - day of this month can be
between 1 to 30 only\n");
                        break;
                case -5:
                        printf("Not valid - this year and month, day can
only be between 1 to 28 only\n");
                        break;
        }

        if (validationCheck==1)
        {
                printf("Valid\n");
        }
    }
}

void validIDs(char stringToValidate[])
{
    char yearString[]={stringToValidate[0],stringToValidate[1]};        /*
a variable that will hold a year of that ID in string format */
    int yearInt;  /* a variable that will hold a year of that ID in integer
format */

    sscanf(yearString,"%d",&yearInt);

    printf("\t\t");

    if (findSpace(stringToValidate))
    {
        printf("Not valid - ID mustn't contain embedded space\n");
    }
```

```
	else if (strlen(stringToValidate)!=11)
	{
		printf("Not valid - ID must be 11 character long\n");
	}
	else if (checkAllDigits(stringToValidate)==0)
	{
		printf("Not valid - ID must be only digit\n");
	}
	else if ((yearInt < 55) || (yearInt > 63))
	{
		printf("Not valid - ID years must be between 55 to 63\n");
	}
	else if (stringToValidate[2]!='0' || stringToValidate[3]!='7' ||
stringToValidate[4]!='0'|| stringToValidate[5]!='5' ||
stringToValidate[6]!='0')
	{
		printf("Not valid - ID 3rd to 7th character must be 07050\n");
	}
	else if (stringToValidate[9]=='0' && stringToValidate[10]=='0')
	{
		printf("Not valid - ID last 2 character mustn't be 00\n");
	}
	else
	{
		printf("Valid\n");
	}
}

void validIdentifiers(char stringToValidate[])
{
	printf("\t\t");

	if (findSpace(stringToValidate))
	{
		printf("Not valid - C Identifier mustn't contain embedded
space\n");
	}
```

```
	else if (checkAllNumberAlphabetUnderscore(stringToValidate)==0)
	{
		printf("Not valid - C Identifier must only contain with alphabet, number or underscore\n");
	}
	else if (stringToValidate[0]!='_' && !isalpha(stringToValidate[0]))
	{
		printf("Not valid - C Identifier must start with alphabet or underscore\n");
	}
	else
	{
		printf("Valid\n");
	}
}

void validPhones(char stringToValidate[])
{
	int currentPositionChecking;  /* a variable that will hold a position that now checking */
	int checkPlusCode;	/* a variable that will hold a status that is plus exist or not */
	int checkCountryCode; /* a variable that will hold a status that country code is valid or not */
	int checkCityCode;	/* a variable that will hold a status that city code is valid or not */
	int checkLastCode;	/* a variable that will hold a status that the last code is valid or not */

	currentPositionChecking=0;
	checkCountryCode=0;
	checkCityCode=0;
	checkLastCode=0;
	checkPlusCode=1;

	printf("\t\t");
```

```
	if (stringToValidate[0]!='+')
	{
		printf("Not valid - Must start with plus\n");
		checkPlusCode=0;
	}
	else if (checkAllNumberDashPlus(stringToValidate)==0)
	{
		printf("Not valid - Must contain with only digit number, 2 dash
and 1 plus\n");
	}
	else
	{
		if (checkPlusCode==1)
		{
			++currentPositionChecking;
			checkCountryCode =
checkPhoneNumberCodeBySection(stringToValidate,currentPositionCh
ecking,1,3,'-');
			currentPositionChecking+=checkCountryCode;
		}
		if (checkPlusCode==1 && checkCountryCode>0)
		{
			++currentPositionChecking;
			checkCityCode =
checkPhoneNumberCodeBySection(stringToValidate,currentPositionCh
ecking,1,3,'-');
			currentPositionChecking+=checkCityCode;
		}
		if (checkPlusCode==1 && checkCountryCode>0 &&
checkCityCode>0)
		{
			++currentPositionChecking;
			checkLastCode =
checkPhoneNumberCodeBySection(stringToValidate,currentPositionCh
ecking,6,10,'\0');
			currentPositionChecking+=checkLastCode;
		}
```

```
		//printf("checkCountryCode is %d\nCheckCityCode is
%d\nCheckLastCode is %d\n",
checkCountryCode,checkCityCode,checkLastCode);
		if (checkCountryCode==-1 || checkCountryCode==-2)
		{
			printf("Not valid - the Country Code must be between 1
to 3 digit\n");
		}
		else if (checkCityCode==-1 || checkCityCode==-2)
		{
			printf("Not valid - the City Code must be between 1 to 3
digit\n");
		}
		else if (checkLastCode==-1 || checkLastCode==-2)
		{
			printf("Not valid - the Last Code must be between 6 to
10 digit\n");
		}
		else
		{
			printf("Valid\n");
		}
	}



}

int findSpace(char stringToFind[])
{
	int spaceChecker;	/* a variable that will hold status that string
is contain space or not */
	spaceChecker=0;

	for (int i = 0; i < strlen(stringToFind); ++i)
	{
		if (stringToFind[i]==BACKSPACE)
```

```
            {
                spaceChecker=1;
            }
        }

        return spaceChecker;
}

int checkAllDigits(char stringToCheck[])
{
        int digitChecker;  /* a variable that will hold status that string is
contain with all digit or not */
        digitChecker=1;

        for (int i = 0; i < strlen(stringToCheck); ++i)
        {
            if (!isdigit(stringToCheck[i]))
            {
                digitChecker=0;
                break;
            }
        }

        return digitChecker;
}

int checkAllNumberAlphabetUnderscore(char stringToCheck[])
{
        int checker;/* a variable that will hold status that string is contain
only number, alphabet and underscore or not */
        checker=1;

        for (int i = 0; i < strlen(stringToCheck); ++i)
        {
            if(!isdigit(stringToCheck[i]) && !isalpha(stringToCheck[i]) &&
stringToCheck[i]!='_')
            {
```

```
                checker=0;
            }
        }

        return checker;
}

int checkAllNumberDashPlus(char stringToCheck[])
{
        int checker;/* a variable that will hold status that string is contain
only number,dash and plus or not */
        int plusQuota;      /* a variable that will be quota for plus sign in that
string */
        int dashQuota;     /* a variable that will be quota for dash sign in
that string */

        checker=1;
        plusQuota=1;
        dashQuota=2;

        for (int i = 0; i < strlen(stringToCheck); ++i)
        {
            if (stringToCheck[i]=='-')
            {
                --dashQuota;
            }
            else if (stringToCheck[i]=='+')
            {
                --plusQuota;
            }
            if(!isdigit(stringToCheck[i]) && stringToCheck[i]!='-' &&
stringToCheck[i]!='+')
            {
                checker=0;
            }
        }
```

```
	if (dashQuota!=0 || plusQuota!=0)
	{
		checker=0;
	}

	return checker;
}

int checkPhoneNumberCodeBySection(char stringToCheck[],int
startPosition,int minSizeOfSection,int maxSizeOfSection,char
expectedEndingCharacter)
{
	int returnCodeStatus;   /* a variable that hold a status of that string
is valid or not */
	int actualSectionSize;   /* a variable that will count a size of string
section that currently check */

	returnCodeStatus=0;
	actualSectionSize=0;

	for (int i = startPosition; i < startPosition+maxSizeOfSection; ++i)
	{
		if (!isdigit(stringToCheck[i]) &&
actualSectionSize<minSizeOfSection)
		{
			returnCodeStatus = -1;
			break;
		}
		else if (stringToCheck[i]==expectedEndingCharacter)
		{
			break;
		}
		else if (isdigit(stringToCheck[i]))
		{
			++actualSectionSize;
		}
	}
```

```
if(stringToCheck[startPosition+actualSectionSize]!=expectedEndingChar
acter && actualSectionSize>=maxSizeOfSection)
	{
		returnCodeStatus = -2;
	}
	//printf("returnCodeStatus is %d\tactualSectionSize is %d\n",
returnCodeStatus,actualSectionSize);
	if(actualSectionSize > 0 && !(returnCodeStatus==-1 ||
returnCodeStatus==-2))
	{
		returnCodeStatus = actualSectionSize;
	}

	return returnCodeStatus;
}

int checkAllNumberUpperAlphabetSpace(char stringToCheck[])
{
	int checker;/* a variable that will hold status that string is valid or
not */
	checker=1;

	for (int i = 0; i < strlen(stringToCheck); ++i)
	{
		if(!isdigit(stringToCheck[i]) && !isupper(stringToCheck[i]) &&
stringToCheck[i]!=' ')
		{
			checker=0;
		}
	}

	return checker;
}

int checkDaySection(char stringToCheck[])
{
```

```
	int returnCodeStatus;   /* a variable that will hold status that a
string is valid or not */
	returnCodeStatus=0;

	for (int i = 0; i < 2; ++i)
	{
		if (isupper(stringToCheck[i]))
		{
			returnCodeStatus=-1;
		}
		else if (stringToCheck[i]==' ')
		{
			returnCodeStatus=-2;
		}
	}

	if (returnCodeStatus==0 && isdigit(stringToCheck[2]))
	{
		returnCodeStatus=-2;
	}
	if (returnCodeStatus==0 && isupper(stringToCheck[2]))
	{
		returnCodeStatus=-3;
	}

	return returnCodeStatus;
}

int checkMonthSection(char stringToCheck[])
{
	int returnCodeStatus;   /* a variable that will hold status that string
is valid or not */
	returnCodeStatus=0;

	for (int i = 3; i < 6; ++i)
	{
		if (isdigit(stringToCheck[i]))
```

```
		{
			returnCodeStatus=-1;
		}
		else if (stringToCheck[i]==' ')
		{
			returnCodeStatus=-2;
		}
	}

	if (returnCodeStatus==0 && isupper(stringToCheck[6]))
	{
		returnCodeStatus=-2;
	}
	if (returnCodeStatus==0 && isdigit(stringToCheck[6]))
	{
		returnCodeStatus=-3;
	}

	return returnCodeStatus;
}

int checkYearSection(char stringToCheck[])
{
	int returnCodeStatus;   /* a variable that will hold status that string
is valid or not */
	returnCodeStatus=0;

	for (int i = 7; i < 11; ++i)
	{
		if (isupper(stringToCheck[i]))
		{
			returnCodeStatus=-1;
		}
		else if (stringToCheck[i]==0)
		{
			returnCodeStatus=-2;
		}
```

```
	}

	if (stringToCheck[11]!=0)
	{
		returnCodeStatus=-2;
	}

	return returnCodeStatus;
}

int dateInputFormatValidation(char stringToValidate[])
{
	int returnCodeStatus;	/* a variable that will hold status that string
is valid or not */
	returnCodeStatus=1;	/* no error */
	if (checkAllNumberUpperAlphabetSpace(stringToValidate)==0)
	{
		returnCodeStatus=-1;	/* error "Not valid - Dates must
contain only with digit number, upper case alphabet and space\n" */	}
	else
	{
		if (checkDaySection(stringToValidate)==-1)
		{
			returnCodeStatus=-2;	/* error "Not valid - Day must
contain only with digit number\n" */
		}
		else if (checkDaySection(stringToValidate)==-2)
		{
			returnCodeStatus=-3;	/* error "Not valid - Day must be
2 digit long\n" */
		}
		else if (checkDaySection(stringToValidate)==-3)
		{
			returnCodeStatus=-4;	/* errror "Not valid - After the
day must be space as in form of (dd MMM yyyy)\n" */
		}
		else
```

```
		{
			if (checkMonthSection(stringToValidate)==-1)
			{
				returnCodeStatus=-5;   /* error "Not valid - Month must contain only with upper case alphabet\n" */
			}
			else if (checkMonthSection(stringToValidate)==-2)
			{
				returnCodeStatus=-6;   /* error "Not valid - Month must be 3 character long\n" */
			}
			else if (checkMonthSection(stringToValidate)==-3)
			{
				returnCodeStatus=-7;   /* error "Not valid - After the month must be space as in form of (dd MMM yyyy)\n" */
			}
			else
			{
				if (checkYearSection(stringToValidate)==-1)
				{
					returnCodeStatus=-8;   /* error "Not valid - Year must contain only with digit number\n" */
				}
				else if (checkYearSection(stringToValidate)==-2)
				{
					returnCodeStatus=-9;   /*error "Not valid - Year must be 4 digit long\n"  */
				}
			}
		}
	}

	return returnCodeStatus;
}

int dateValidationCheck(char stringToValidate[])
{
```

```
	int returnCodeStatus;	/* a variable that will hold status that string is valid or not */
	char monthChecker[12][4]={"JAN","FEB","MAR","APR","MAY","JUN","JUL","AUG","SEP","OCT","NOV","DEC"};  /*  */
	int monthInt;		/* a variable to hold month in integer format */
	int day;	/* a variable to hold day in onteger format */
	char month[4];	/* a variable to hold month in string format */
	int year;	/* a variable to hold year in integer format */
	returnCodeStatus=1;
	monthInt=0;

	sscanf(stringToValidate,"%d %s %d",&day,month,&year);
	//printf("%d\n%s\n%d\n", day,month,year);
	if (year>2120 || year<1920)
	{
		returnCodeStatus=-1;	/* error "Not valid - year can't be more than 100 year in the pass or the future\n" */
	}
	else
	{
		for (int i = 0; i < 12; ++i)
		{
			if (strcmp(month,monthChecker[i])==0)
			{
				monthInt=i+1;
				break;
			}
		}

		if (monthInt==0)
		{
			returnCodeStatus=-2;	/* error "Not valid - month need to be upper case of first 3 character only\n" */
		}
		else if (monthInt==1 || monthInt==3 || monthInt==5 || monthInt==7 || monthInt==8 || monthInt==10 || monthInt==12)
```

```
		{
			if (day >31)
			{
				returnCodeStatus=-3;	/* error "Not valid - day of this month can be between 1 to 31 only\n" */
			}
		}
		else if (monthInt==4 || monthInt==6 || monthInt==9 || monthInt==11)
		{
			if (day>30)
			{
				returnCodeStatus=-4;	/* error "Not valid - day of this month can be between 1 to 30 only\n" */
			}
		}
		else
		{
			if (day>29)
			{
				returnCodeStatus=-5;	/* error "Not valid - this year and month, day can only be between 1 to 28 only\n" */
			}
			else if (day==29 && (year%4!=0 || year%400!=0 && year%100==0))
			{
				returnCodeStatus=-5;	/* error "Not valid - this year and month, day can only be between 1 to 28 only\n" */
			}
		}
	}

	return returnCodeStatus;
}

/*****************************************
```

```
fgets(terminalInput,sizeof(terminalInput),stdin);
sscanf(terminalInput,"%",&);

**********************************************
*/
```