C Supplementary

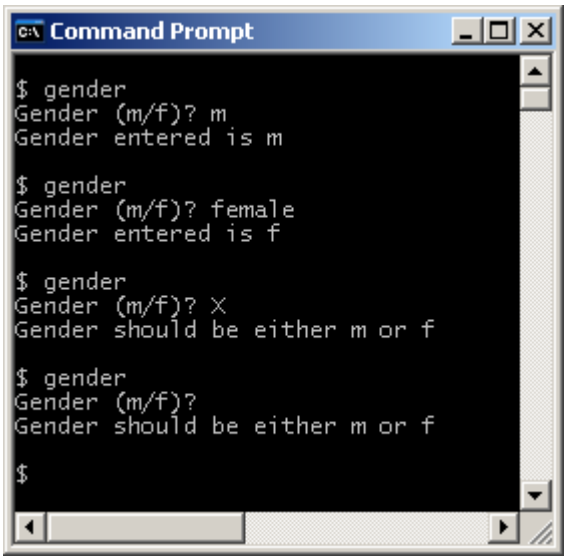
Terry Marris February 2013

Validation

Validation involves checking that data falls within pre-defined limits. For example, a person's age must be between 0 and 125. A person's gender must be either *f* or *F* or *m* or *M*. A date's day/month/year combination must be consistent. We take a look at validation.

Gender

A person's gender, male or female, is sometimes asked for in forms. Valid input would be *m*, *M* , *f*, or *F*. Invalid input would include blank space or *X*. A program run is shown below.



The essential principle is that the function that gets the input returns the error, if any, to the caller, and it is up to the caller to deal with the error.

First, we define values for *OK* and the error state, *ERR_GENDER*.

```
#define OK 0
#define ERR_GENDER 1
```

You could use any numeric values for *OK* and *ERR_GENDER* you like, just as long as they are different from each other.

As usual, we read a string from the keyboard, and convert the input to the required type. In our case, the required type is just a single *char*.

```
/* getGender: reads m or f into gender.
   Returns OK if successful, or ERR_GENDER if an error occurs. */
int getGender(char *gender)
{
  printf("Gender (m/f)? ");
  getString(gender);
  gender[0] = tolower(gender[0]);
  if (gender[0] == 'f' || gender[0] == 'm')
    return OK;
  return ERR_GENDER;
}
```

getString() is the function that reads the string input from the keyboard. It has been changed slightly from the previous version. *maxLength* is no longer a parameter (thereby reducing coupling). Its value is calculated from the array of *char* passed to the function.

```
int getString(char string[])
{
  ...
  int maxLength = sizeof(string) / sizeof(char);
  ...
```

The rest of the function remains unchanged from before and it is presented here without further comment.

```
/* getString: reads characters, plus '\0', into string from keyboard.
   Returns number of chars (excluding '\0') read. */
int getString(char string[])
{
  char c;
  int i = 0;
  int maxLength = sizeof(string) / sizeof(char);

  while ((c = getchar()) != '\n') {
    if (i < maxLength - 1) {
      string[i] = c;
      i++;
    }
  }
  string[i] = '\0';
  return i;
}
```

getGender() is called in *main()* and, therefore, it is *main()*'s responsibility to deal with any error value returned.

```
int main()
{
  char gender[2] = { '\0' };
  int report;

  report = getGender(gender);
  if (report == OK)
    printf("Gender entered is %c\n", gender[0]);
  error(report);
  return 0;
}
```

main() deals with any error returned from *getGender()* by calling *error()*.

```
/* error: displays report. */
int error(int report)
{
  switch (report) {
  case OK:
    return 0;
  case ERR_GENDER:
    printf("Gender should be either m or f\n");
    return 0;
  default:
    printf("Unexpected error\n");
  }
  return 0;
}
```

In *error()*, *OK* is ignored, but, otherwise, an informative error message is displayed.

Here is the complete program.

```
#include <stdio.h>
#include <ctype.h>

#define OK 0
#define ERR_GENDER 1

/* error: displays report. */
int error(int report)
{
  switch (report) {
  case OK:
    return 0;
  case ERR_GENDER:
    printf("Gender should be either m or f\n");
    return 0;
  default:
    printf("Unexpected error\n");
  }
  return 0;
}


/* getString: reads characters, plus '\0', into string from keyboard.
   Returns number of chars (excluding '\0') read. */
int getString(char string[])
{
  char c;
  int i = 0;
  int maxLength = sizeof(string) / sizeof(char);

  while ((c = getchar()) != '\n') {
    if (i < maxLength - 1) {
      string[i] = c;
      i++;
    }
  }
  string[i] = '\0';
  return i;
}
```

```
/* getGender: reads m or f into gender.
   Returns OK if successful, or ERR_GENDER if an error occurs. */
int getGender(char *gender)
{
  printf("Gender (m/f)? ");
  getString(gender);
  gender[0] = tolower(gender[0]);
  if (gender[0] == 'f' || gender[0] == 'm')
    return OK;
  return ERR_GENDER;
}


int main()
{
  char gender[2] = { '\0' };
  int report;

  report = getGender(gender);
  if (report == OK)
    printf("Gender entered is %c\n", gender[0]);
  error(report);
  return 0;
}
```

Exercise

1 Write and test a program that inputs a person's age and determines whether the age input is valid. A valid age is between 0 and 125 inclusive.

2 Validating a date is a traditional programming exercise. A valid day may be between 1 and 31 if the month is 1, 3, 5, 7, 8, 10 and 12, between 1 and 30 if the month is 4, 6, 9, 11, (30 days hath September, April, June and November, ...) and between 1 and 28 if the year is a none-leap year, between 1 and 29 if the year is a leap year. You may remember a year is a leap year if it is exactly divisible by 4, except that years divisible by 100 are not leap years, but years divisible by 400 are.