Série 12 - Les Structures et structure avec champs de bits

But

In this Série you will learn how to define and use structures. Moreover, by working on the optional exercise, you will have a better understanding of the bitfields.

Exercice 1 (document séparé / niveau 0): Définition de structures (en anglais)

Exercice 2 (niveau 1): Structure et fonction

L’exercice 1 a montré le passage par valeur de deux structures de type CIRCLE à la fonction circleIntersect(). Nous désirons ici définir deux variantes d’une fonction destinée à lire une structure pour appliquer le principe d’abstraction au programme de l’exercice 1.

2.1) La première variante de la fonction a pour déclaration :

CIRCLE circleRead (void) ;

Son but est de lire les données d’une seule variable de type CIRCLE ; cette variable est locale à la fonction circleRead(). La fonction circleRead() utilise return pour renvoyer la valeur de la variable initialisées. Reprenez le code de lecture et de vérification disponible dans l’exercice 1 pour définir la fonction circleread(). Ensuite créez une fonction main() qui déclare et initialise une variable locale de type CIRCLE comme suit :

int main(void)
{
    CIRCLE cercle = circleRead() ;

    // puis affichez chaque champs de cercle avec printf()
    ....
}

2.2) La seconde variante de la fonction a pour déclaration :

void circleReadp (CIRCLE *p) ;

Son but est de lire les données d’une seule variable de type CIRCLE grâce à l’adresse fournie en paramètre. Vérifiez que p est différent de NULL avant de l’utiliser.

Créez une fonction main() qui utilise votre fonction (sans planter)
int main(void)
{
    CIRCLE cercle = {0.,{0.,0.}};
    CIRCLE *pcercle=NULL;

    circleReadp(&cercle);
    circleReadp(pcercle);

    // puis affichez chaque champs de cercle avec printf()
    ....
}

2.3) Toujours pour appliquer le principe d'abstraction, définissez une fonction qui affiche la valeur de chaque d'une structure de type CIRCLE qu'on lui fournit en paramètre. Utilisez cette fonction pour remplacer les printf() des questions 2.1 et 2.2 par un seul appel à cette nouvelle fonction.

Exercice 3 (niveau 2, en anglais, optional exercise): Structure avec champs de bits (bitfields)

In this exercise you are supposed to develop a program to automatically control a lunar robot. The robot is basically a vehicle with a camera moving on the lunar surface. It captures a video of the path it visits and takes a high resolution photo whose information is given. Your duty is to program the robot’s decision unit which reads data from the peripheral devices, i.e. position, temperature etc., evaluates it, and sends necessary commands to the robot.

In order to simplify the problem, we assume that the lunar surface is planar. You can assume everything as simple as possible.

Data is transferred between the decision unit and the peripheral devices with predefined packet formats using struct.

The data is received from peripherals via INCOMING_DATA data structure and data is sent via OUTGOING_DATA data structure.

INCOMING_DATA has four different fields:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>position</td>
<td>POSITION: (float x, float y)</td>
<td>A struct holding the current position (x, y) of the robot.</td>
</tr>
<tr>
<td>temperature</td>
<td>integer</td>
<td>Temperature of the engine whose range is [-125, 125] °C.</td>
</tr>
<tr>
<td>illuminance</td>
<td>unsigned integer</td>
<td>A parameter which measures how dark the environment is. Zero represents the darkest and ten represents the brightest condition.</td>
</tr>
<tr>
<td>error_code</td>
<td>unsigned integer</td>
<td>There are 59 different errors whose IDs vary from 1 to 59.</td>
</tr>
</tbody>
</table>
OUTGOING_DATA has 12 different fields:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>engine</td>
<td>unsigned integer</td>
<td>A parameter whose value is one while the engine is activated and zero if it is not activated.</td>
</tr>
<tr>
<td>id</td>
<td>unsigned integer</td>
<td>We set a unique id for each command we send. We simply set it as the number of the data already sent. It can be any unsigned integer.</td>
</tr>
<tr>
<td>velocity_sign_part</td>
<td>unsigned integer</td>
<td>Sign of the velocity. It is either zero or one. If it is zero it means that velocity is positive and if it is one the velocity is negative.</td>
</tr>
<tr>
<td>velocity_decimal_part</td>
<td>unsigned integer</td>
<td>Integer part of the decimal number representing the velocity component which is along the head of the robot. Its range is [0, 30].</td>
</tr>
<tr>
<td>velocity_fractional_part</td>
<td>unsigned integer</td>
<td>Fractional part of the decimal number representing the velocity component which is along the head of the robot. Its range is [0, 9].</td>
</tr>
<tr>
<td>angular_velocity_sign_part</td>
<td>unsigned integer</td>
<td>Sign of the angular velocity. It is either zero or one. If it is zero it means that velocity is positive and if it is one the velocity is negative.</td>
</tr>
<tr>
<td>angular_velocity_decimal_part</td>
<td>unsigned integer</td>
<td>Integer part of the decimal number representing the turning velocity component. Its range is [0, 30].</td>
</tr>
<tr>
<td>angular_velocity_fractional_part</td>
<td>unsigned integer</td>
<td>Fractional part of the decimal number representing the turning velocity component. Its range is [0, 9].</td>
</tr>
<tr>
<td>headlight</td>
<td>unsigned integer</td>
<td>A parameter whose value is one while the headlight is activated and zero if it is not activated.</td>
</tr>
<tr>
<td>cooler</td>
<td>unsigned integer</td>
<td>A parameter whose value is one while the cooler is activated and zero if it is not activated.</td>
</tr>
<tr>
<td>camera_height</td>
<td>unsigned integer</td>
<td>It controls the height of the camera. Its range is [0, 50] cm where 0 is the lowest position and 50 is the highest one.</td>
</tr>
<tr>
<td>take_photo</td>
<td>unsigned integer</td>
<td>A parameter whose value is one to capture a photo and zero, otherwise.</td>
</tr>
</tbody>
</table>

As you might notice, for representing the velocity and the angular velocity, we divide a number into three parts: a sign part, a decimal part and the fractional part. For instance, integer part of 10.7 is 10 and its fractional part is 7. Since it is positive, sign should be set to zero. Therefore, numbers represented with this system would be: -30.9, -30.8... 30.7, 30.8, 30.9.

- What is the minimum number of bits that we need for each field of OUTGOING_DATA?
- Now, considering the size of each field, implement a struct by using bit fields. First, keep the order given in the question when defining it. Use `sizeof` operator to estimate its total size.
- We can lower the size of OUTGOING_DATA by considering padding and alignment rules of the C language. Now, redesign your struct in a way to obtain a lower packet size. Use `sizeof` operator and observe the result.
- Is it possible to determine the size of a struct without using `sizeof`? How?

Now, let’s write the basic control mechanism. Open “main.c” file. For this exercise, you only need to change the content of the main function. The parts you need to modify are already noted as “MODIFY IT!”. You do not need to modify other parts or files. You will find a “robot.h” file where OUTGOING_DATA and INCOMING_DATA are already defined. Therefore, you do not need to make
any definitions about them, either. Moreover, you will be given “robot.o” where the definitions of
the functions declared in “robot.h” are compiled into. Use simply

    gcc main.c robot.o -o command_robot

cmd command to compile your program. In addition, you can add some compilation flags such as –Wall if
required.

We have very simple rules for controlling the robot:

- First of all, we need to start the engine. To do that, set engine to one. engine must stay one
  throughout the mission.
- Operation of the robot will be implemented in an iterative way. You must first call
  get_total_step_number function to get how many steps, iterations in your program, this
  mission will take. After then, get_photo_step_number is called to get at which step, iteration
  number, the photo should be taken.
- After then we enter the iteration phase:
  o First, INCOMING_DATA is read with get_data function.
  o Set the id of the command as the number of the current step.
  o Positional data received with INCOMING_DATA is passed to decide_velocity function.
    It returns the velocity value we need to set to the robot.
  o Positional data received with INCOMING_DATA is passed to decide_angular_velocity
    function. It returns the angular velocity value we need to set to the robot.
  o Positional data received with INCOMING_DATA is passed to decide_camera_height
    function. It returns the camera height value we need to set to the robot.
  o If the illuminance value is less than 2, the robot needs to switch on the headlight. If it
    is greater than 4 it should switch off the headlight.
  o If the temperature is greater than 75°C, the cooler should be switched on. When the
    temperature lowers to less than 40°C it should be switched off.
  o If the step number is equal to photo_step_number, robot must take a photo.
  o Pack all these commands into OUTGOING_DATA and call set_data function with it.
- When the total_step_number is reached, the engine must be switched off. And the
  execution ends.

Please note that, get_total_step_number, get_photo_step_number, get_data, decide_velocity,
decide_angular_velocity, decide_camera_height, set_data functions are all implemented and their
declarations can be found in “robot.h”.

Also, please note that whenever you call set_data, you will be provided with some feedback.

- If you do not set the velocity properly, for instance, a warning will appear: “Velocity was set a
  wrong value!”.
- Similarly, for angular velocity, you will be warned: “Angular velocity was set a wrong value!”.
- If you do not switch on the headlight whereas it is needed: “Did you forget to switch on the
  headlight?”.
- If you switch on the headlight before, but, you do not need it anymore, and you did not
  switch it off: “Did you forget to switch off the headlight?”. 
• If the cooler is needed and you do not start it: “Did you forget to switch on the cooler?”.
• If you forget to switch it off when needed: “Did you forget to switch off the cooler?”.
• If you do not set the height of the camera to the expected value: “Camera height was set a wrong value!”.
• If you do not take the photo at the expected step: “Photo has been taken at the wrong step!”
• If you take the photo at the right step: “Photo has been taken successfully!”.

In the end of the execution of your code, you will be given either “MISSION ACCOMPLISHED SUCCESSFULLY!” or “MISSION FAILED! CHECK ERROR MESSAGES TO FIND YOUR ERROR!” according to your success.

Hint: If you use output redirection, you can analyze the error messages easily:

```
./command_robot > my_output.txt
```

Hint2: Please note that you need to deal with the precision issue when converting the decide_velocity function’s return value, which is a float, into our number format. For instance, if the velocity is supposed to set as 19.0, the value you obtain can be like 18.9999. A simple solution to handle rounding properly is to add 0.05 to the velocity and angular velocity values that you get and then to do the conversion.