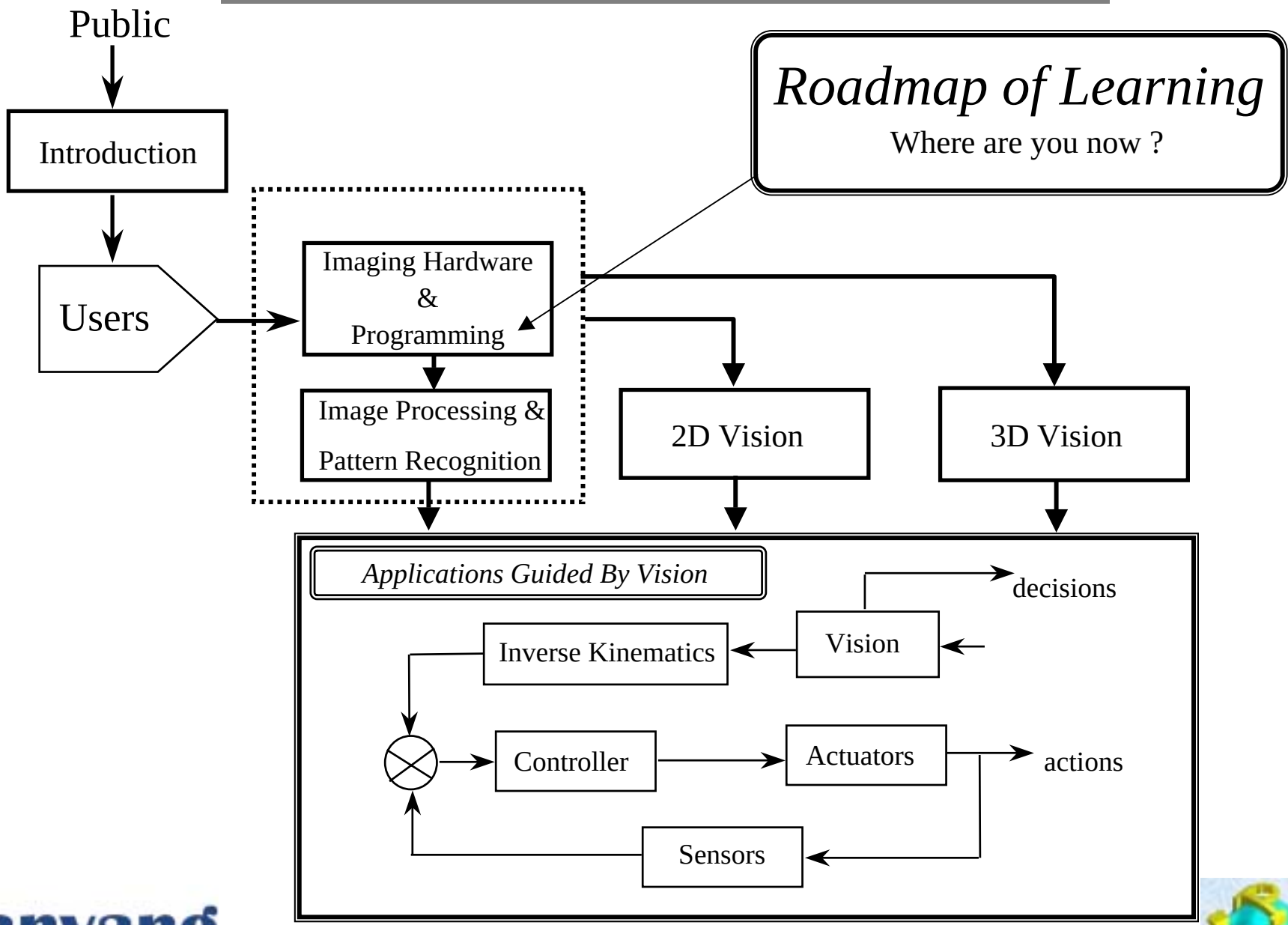
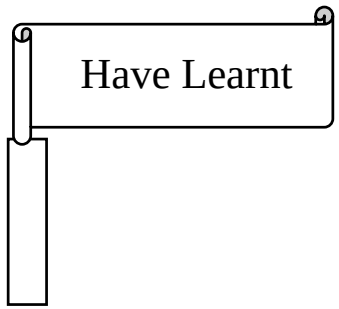
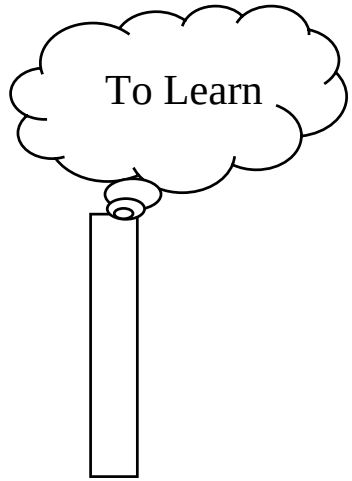


CONTENT

Chapter 3: Fundamentals of Programming





3.1 Basics of C programming

3.2 Data Structures in C

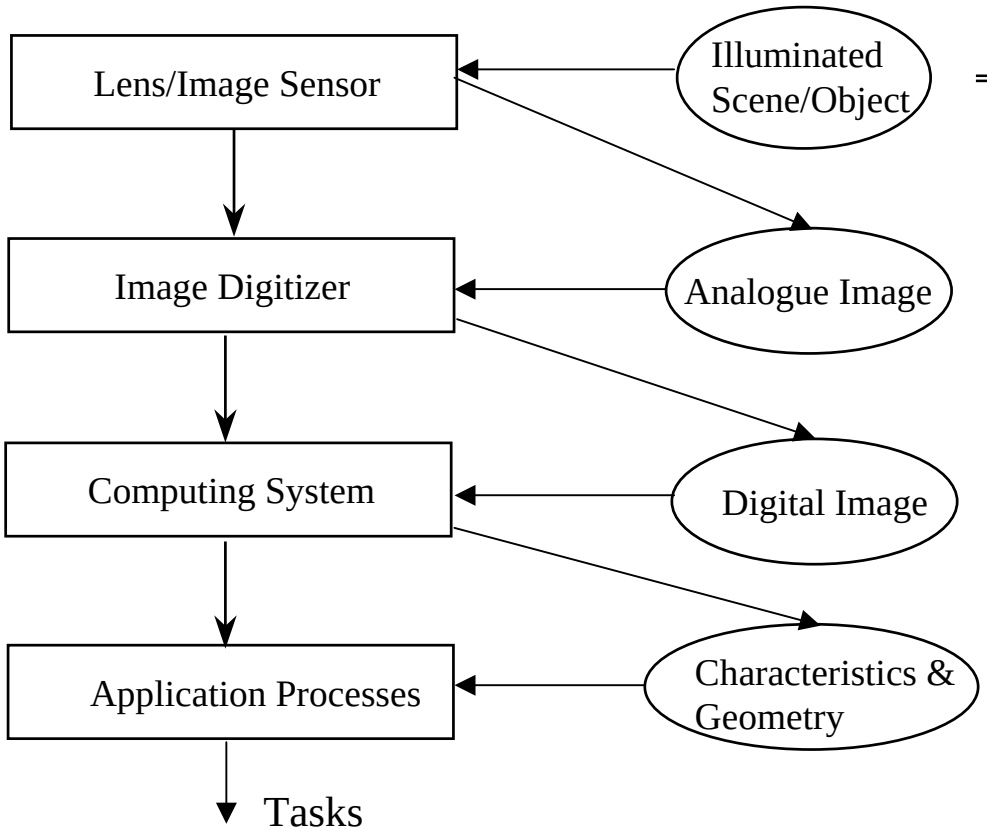
3.3 Data Processing Loops in C

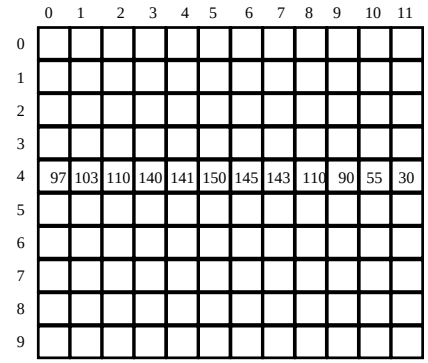
3.4 Use of Image and Graphics Library

3.5 Memory Concept

What is a machine vision system ? (A Review)

ANSWER:





Question:

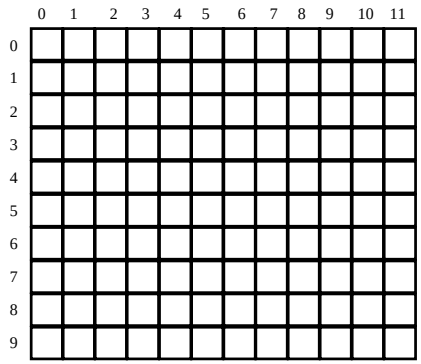
How to program a machine vision application ?

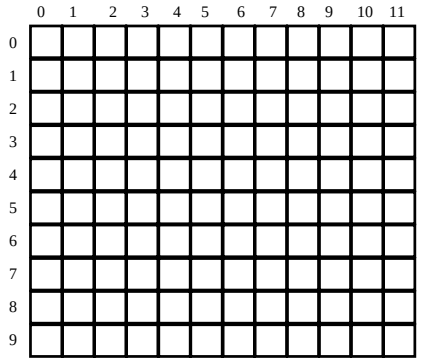
What is a digital image from the viewpoint of a C programmer ?

ANSWER:

Two dimensional matrix !

If a pixel value is coded in 8 bits (one Byte), then a digital image is a 2D matrix of numbers of the type of “unsigned char”.





toto.c

```
unsigned char        left_image[512][512] ;
int                  resolution_x_of_left_image ;
int                  resolution_y_of_left_image ;

unsigned char        right_image[512][512] ;
int                  resolution_x_of_right_image ;
int                  resolution_y_of_right_image ;

main(int argc, char **argv)
{
      /* read in images  */

     /* process images */

     /* display images */
}
```


From this example, we notice that variables can be grouped into a category so as to make programming easier and more readable.

How to do so ?

ANSWER: To use data structure !

toto.c

```
unsigned char       left_image[512][512] ;
int                 resolution_x_of_left_image ;
int                 resolution_y_of_left_image ;

unsigned char       right_image[512][512] ;
int                 resolution_x_of_right_image ;
int                 resolution_y_of_right_image ;

main(int argc, char **argv)
{
      /* read in images  */

     /* process images */

     /* display images */
}
```


In C, the command “typedef struct” is to define a new type of variable that is a data structure (ie,: a collection of variables of different types).

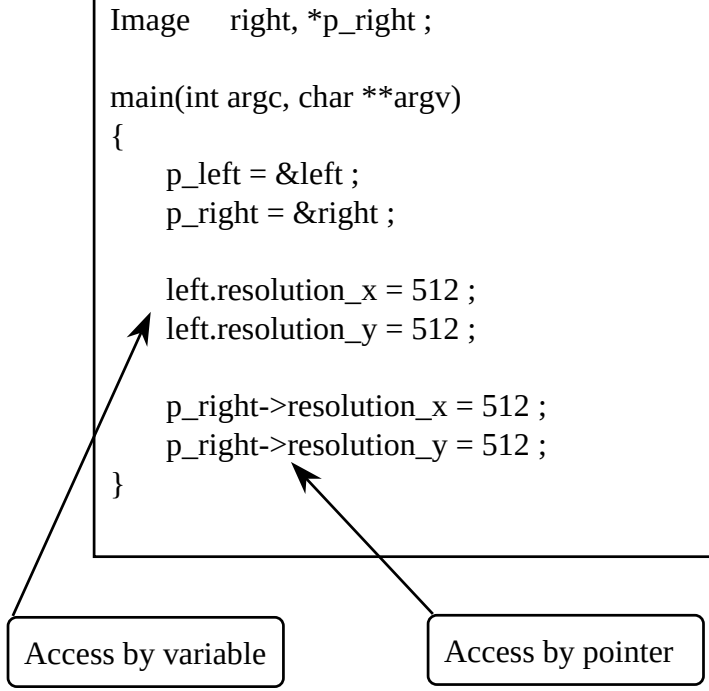
The elements of a data structure can be accessed in two ways:

1. [variable of data structure].[name of element]

2. [pointer of data structure]->[name of element]

toto.c

```
#include <stdio.h>

typedef struct
{
      unsigned char      pixmap[512][512] ;
      int                resolution_x ;
      int                resolution_y ;
}     Image ;

Image     left, *p_left;
Image     right, *p_right ;

main(int argc, char **argv)
{
      p_left = &left ;
      p_right = &right ;

      left.resolution_x = 512 ;
      left.resolution_y = 512 ;

      p_right->resolution_x = 512 ;
      p_right->resolution_y = 512 ;
}
```



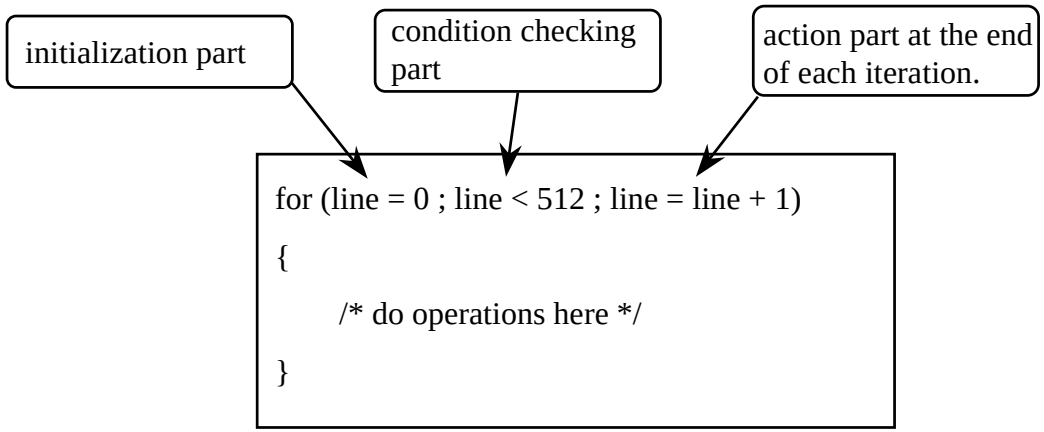
To process a digital image that is a two dimensional matrix, one needs to scan line by line and column by column.

How to do this in C ?

ANSWER: To use instructions for loop control !

	0	1	2	3	4	5	6	7	8	9	10	11
0												
1												
2												
3												
4	97	103	110	140	141	150	145	143	110	90	55	30
5												
6												
7												
8												
9												

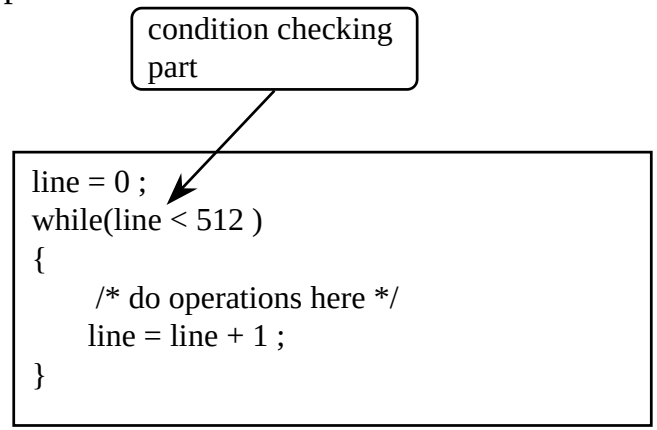
Case 1: “for(…)” loop:



toto.c

```
#include <stdio.h>


main(int argc, char **argv)
{
      int    line ;

      for (line = 0 ; line < 512 ; line = line + 1)
      {
            printf(\n Now, we are at line: %d”, line) ;
      }
}
```


Case 2: "while(…)" loop:



toto.c

```
#include <stdio.h>


main(int argc, char **argv)
{
      int    line ;
      line = 0 ;
      while(line < 512 )
      {
            printf(\n Now, we are at line: %d", line) ;
            line = line + 1 ;
      }
}
```


Exercise 1

To set the pixel value “0” to all the pixels in the left image, and “1” to all the pixels in the right image.

toto.c

```
#include <stdio.h>

typedef  unsigned char  Byte ;

typedef struct
{
        Byte       pixmap[512][512] ;
        int        resolution_x ;
        int        resolution_y ;
}       Image ;

static  Image     left, right ;

main(int argc, char **argv)
{
        int     line, column ;

        /* set 0 to all the pixels in the left image */
        for (line = 0 ; line < 512 ; line = line + 1)
        {
                for (column = 0 ; column < 512 ; column++)
                {
                        left.pixmap[line][column] = 0 ;
                }
        }


        /* set 1 to all the pixels in the right image */
        for (line = 0 ; line < 512 ; line++)
        {
                for (column = 0 ; column < 512 ; column++)
                {
                        right.pixmap[line][column] = 1 ;
                }
         }
}
```

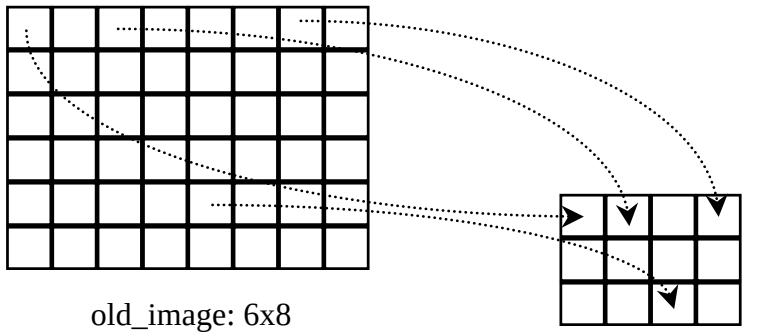

Exercise 2

To create a new image by reducing an old image into the half of its initial size.

Mathematics:

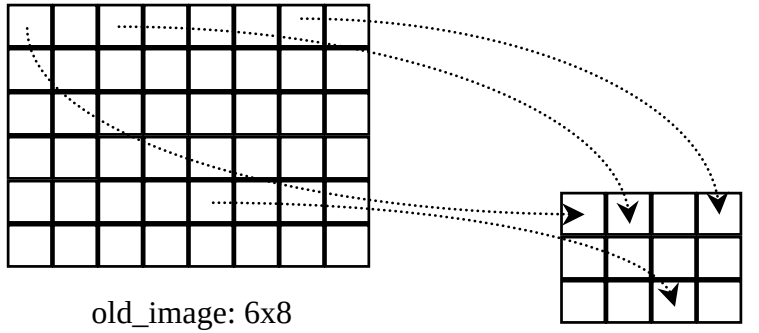
if image1 is the reduced version of image2 by half, then:

image1[line][column] = image2[2*line][2*column];



old_image: 6x8

new_image: 3x4



old_image: 6x8

new_image: 3x4

toto.c

```
#include <stdio.h>

typedef  unsigned char   Byte ;

static  Byte    old_image[512][512] ;
static  Byte    new_image[256][256] ;

static  void    ReadInImage()
{

}

main(int argc, char **argv)
{
        int    line, column ;

        ReadInImage() ;

        for (line = 0 ; line < 256; line++)
        {
                for (column = 0 ; column < 256 ; column++)
                {
                        new_image[line][column] = old_image[2*line][2*column];
                }
        }
}
```


Results:

Exercise 3

If a pixel value is coded in 8 bits (one Byte), it will vary from 0 to 255 (ie,: 0000 0000 to 1111 1111). Given an image of the size of 512x512, compute the number of occurrence for each pixel value in [0, 255].

101							
		101		101		66	
		66			101		
			101	66			

The number of occurrence for “66” is “3”;

The number of occurrence for “101” is “5”.

toto.c

```
#include <stdio.h>

typedef unsigned char  Byte ;

static Byte    one_image[512][512] ;
static int     o_number[256] ;

static  void   ReadInImage()
{

}

main(int argc, char **argv)
{
        int   line, column, pixel_value ;

        ReadInImage() ;

        for (pixel_value = 0 ; pixel_value < 256 ; pixel_value++)
                o_number[pixel_value] = 0 ;

        for (line = 0 ; line < 512 ; line++)
        {
                for (column = 0 ; column < 512 ; column++)
                {
                        pixel_value = one_image[line][column] ;
                        o_number[pixel_value] = o_number[pixel_value] + 1;
                }
         }
}
```


SUMMARY

1. It is a convenient way to group related variables into a structure.

2. The easiest way to scan a digital image is to use loop controls.

3. In C, the most useful loop controls are:

   * for(…)
   * while(…)

4. “for(…)” loop control has three parts:

   * initialization part
   * condition checking part
   * action part at the end of each iteration

5. “while(…)” only has the “condition checking part”.