École Centrale de Nantes

AlPro

Tutorial 1

1.1 Points and polygon (record)

We want to compute properties of some polygon (perimeter, average edge length, bounding box,...). The bounding box is a rectangle enclosing the polygon with sides parallel to the axes. The vertices of the polygon are given (in the order of the perimeter route) by real coordinates in a plane and the name of the point (char).

The list of coordinate is supposed correct and is only read once to store the polygon at the beginning. An example of list of point is the following :

- 1,5 3,7 A
- 6,4 4,6 B
- 5,8 8,3 C
- ...
- 1,5 3,7 A (last vertex, do not add it to the data structure)

The coordinates and the name of the last vertex are the same as those of the first point. This vertex only allows us to detect the closing of the polygon. It should not be entered into the data structure.

- **Propose a data structure to store the polygon,**
- **Write the main algorithm to read a polygon, compute properties and print them (just by calling secondary functions, without writing the content of these functions),**
- **Write the content of the secondary functions to compute the perimeter, the average edge length, the bounding box.**

```
typedef struct Point{
  Char name;
  Real x;
  Real y;
}Point;

typedef Point[] Polygon;


/* main function for polygons */
function main{

  Polygone poly←readPolygone();
  Real pe←perimeter(poly);
  Real average_l←average_length(poly);
  Polygone box←bounding_box(poly);
  print(...);
}

/* function to compute the size of a polygon */
Integer size(Polygon p){
  if (not p==NULL) return sizeof(p)/sizeof(p[0]);
  return 0;
}


/* function to compute the perimeter of a polygon using a function for the
   distance between two points */
```

```
Real perimeter(Poligon p){
  Real p←0;
  for i from 0 to size(p)−1{ // size(p) is  when it's not empty
    pe← pe + distance(p[(i−1)\%size(p)],p[i]);
  }return pe;
}

/* function to compute the distance between two points */
Real distance(Point a, Point b){
  return √((a.x-b.x)^2 + (a.y-b.y)^2)
}


/* function to compute the average edge length */
Real average_length(Polygon p){
  return perimeter(p)/size(p);
}


/* function to compute the bounding box */
Polygon bounding_box(Polygon p){
  Real xmin←p[0].x,
  Real xmax←p[0].x;
  Real ymin←p[0].y;
  Real ymax←p[0].y;

  for i from 0 to size(p)−1{
    if(p[i].x<xmin) xmin←p[i].x;
    if(p[i].x>xmax) xmax←p[i].x;
    if(p[i].y<ymin) ymin←p[i].y;
    if(p[i].y>ymax) ymax←p[i].y;
  }

  Polygon box[4];
  box[0].x←xmin;
  box[0].y←ymin;
  box[1].x←xmin;
  box[1].y←ymax;
  box[2].x←xmax;
  box[2].y←ymax;
  box[3].x←xmax;
  box[3].y←ymin;

  return box;
}
```

1.2 Queen and pawns

We consider an 8 by 8 chessboard containing 8 pawns and a queen. The queen's position is just its two coordinates and the positions of the pawns will be stored in an 8 by 8 matrix. We will define them in the following :

```
int[8][8] chessBoard; // this is a boolean matrix, with only 0s and 1s

int qx, qy; // the queens coordinates
```

The queen can move on the lines (horizontal, vertical and diagonal) on which she is standing. If two pawns or more are over the same line, only the first is in the capture. For a pawn, to be in the capture means that the queen can capture it in her next move.

— **Write the main algorithm to compute this problem. Try to use only one secondary function nbr_pawns to compute the number of pawns in each line.**
— **Write the content of this function.**

There are different ways to write these functions, here is one, with two different approaches for the secondary function

```
/* main algorithm to get the number of pawns in the capture */
function main{

  Integer nbr ← 0;
  for i from −1 to 1{ // i gives the direction on the x−axis
    for j from −1 to 1{ // j gives the direction on the y−axis
      if (not (i==0 AND j==0)){ // to ignore the position with to the queen on it
        nbr ← nbr + is_pawn(chessBoard, qx, qy, i, j);
        // or equivalently with rec _is_pawn(chessBoard, qx, qy, i, j);
      }
    }
  }
  print("The number of pawns in the capture is: " + nbr);
}



/* function that given a position (qx,qy) and a direction (i,j) visits all the
   squares in the line to see if there is at least one pawn. */

Integer is_pawn(chessBoard, qx, qy, i, j){
  Integer k=1;
  while(0≤qx+k*i≤7 AND 0≤qy+k*j≤7){
    if (chessBoard[qx+k*i][qy+k*j])
      return 1;
  }return 0;
}



/* recursive function that given a position (x,y) and a direction (i,j) visits
   all the squares in the line to see if there is at least one pawn. */

Integer rec_is_pawn(chessBoard, x, y, i, j){
  if( not ((0≤x+k*i≤7 AND 0≤y+k*j≤7))
    return 0;
  if (chessBoard[x+k*i][y+k*j])
      return 1;
  return rec_is_pawn(chessBoard,x+k*i,y+k*j,i,j);
}
```

1.3 Recursion

Compute using a recursive algorithm :
— the maximum of an integer array
— the reverse printing of an integer array
— a function to reverse a word and another to check if the word is a palindrome (a word which reads the same forward and backward, such as madam, radar, level,...)

```
/* recursive maximum of an integer array */

Integer max_rec(array,pos){ // start with pos=0
  if(pos==size(array)−1) return array[pos];
```

```
  max←max_rec(array,max,pos+1);
  if(array[pos]>max) max←array[pos];
  return max;
}


/* recursive reverse printing */

function reverse_print(array,start){ //start with pos=0
  if(start==size(array)){
    printf("\n");
    return;
  }
  reverse_print(array,start+1);
  printf("\%d ",arr[start]);
}

function other_reverse_print(array,start){ //start with pos=size(array)-1
  if(start==-1){u
    printf("\n");
    return;
  }
  printf("\%d ",arr[start]);
  other_reverse_print(array,start-1);
}

/* recursively reverse a word using pointer properties */
CharArray reverse(in,out,size){
  if(size<0)
    return out;
  out[size]=in[0];
  reverse(in+1,out,size-1);
}

/* recursively reverse a word (an array arr of char) */
CharArray reverse(arr,new,current){
  //start with new of same size as arr, current=0
  if(current>(size(arr)-1)) return new;
  new[size(arr)-1-current]←arr[current];
  return reverse(arr,new,current+1);
}

/* test if palindrome */
Boolean palindrome(arr1,arr2,current){ // start with current=0
  if(current>(size(arr)+1)/2) return True;
  if(arr2[size(arr)-1-current] NOT= arr1[current]) return False;
  return palindrome(arr1,arr2,current+1);
}
```