1. Introdução. A seguir encontra-se a implementação em C de um algoritmo para triangularização de polígonos. Este algoritmo tem complexidade de tempo $O(n^3)$, onde n é o número de vértices do polígono.

Note que seria mais natural para este algoritmo se tivessemos representado um polígono como uma lista duplamente ligada de seus vértices. No entanto, para ficar mais fácil e clara a apresentação das idéias do algoritmo, um polígono está sendo representado por um vetor. Para efeito da complexidade assintótica do algoritmo esta escolha não traz problema algum. (Apesar de que o vetor deve ser quase que inteiramente recopiado toda a vez que a função *ClipEar* corta um orelha do polígono—se tivessemos usado um lista duplamente ligada a complexidade de tempo de *ClipEar* seria $O(1)$ ao invés de $O(n)$.)

```
#define X 0
#define Y 1
#define DIM 2      /* dimensão dos pontos */
#define PMAX 1000      /* número máximo de pontos no polígono */
  typedef enum {
    FALSE, TRUE
  } bool;
  typedef int tPointi[DIM];      /* tipo ponto inteiro */
  typedef double tPointd[DIM];      /* tipo ponto real */
  typedef tPointi tPolygoni[PMAX];      /* polígono do tipo inteiro */
  bool Diagonal(int i, int j, int n, tPolygoni P);
```

2. Triangulate. Triangulariza um polígono P encontrando $n-3$ diagonais. Escreve as coordenadas dos pontos extremos das diagonais. Seria preferível que o programa imprimisse o índice dos pontos ao invés de suas coordenadas. Uma manira de consertar isso é manter um vetor com o indice dos vértices. Efeito colateral desta implementação: destroi a estrutura de dados que guarda o polígono P. Uma maneira fácil de evitar esse efeito indesejado é fazer uma cópia do polígono antes de começar a triangularização.

```
void Triangulate(int n, tPolygoni P)
{
  int i, i1, i2;
  if (n > 3)
    for (i = 0; i < n; i++) {
      i1 = (i + 1) % n;
      i2 = (i + 2) % n;
      if (Diagonal(i, i2, n, P)) {
        PrintPoint(P[i]);
        putchar('\t');
        PrintPoint(P[i2]);
        putchar('\n');
        ClipEar(i1, n, P);
        Triangulate(n - 1, P);
        break;
      }
    }
}
```

3. ClipEar. Esta função é implementada copiando-se as posições do $[i+1, n-1]$ do vetor contendo os vértices do polígono para as posições $[i, n-2]$], detruindo portanto o conteúdo da posição i. Isso significa que *ClipEar* tem complexidade de tempo $O(n)$. Este custo poderia ser $O(1)$ (tempo constante) se usassemos um lista ligada para representar o polígono ao invés de um vetor.

```
void ClipEar(int i, int n, tPolygoni P)
{
  int k;
```

```
    for (k = i; k < n − 1; k++) PointAssign(P[k], P[k + 1]);
}
```

4. PointAssign. Implementa $a \leftarrow b$, atribuição de pontos/vetores. Atribuição entre vetores não é possível em C.

```
void PointAssign(tPointi a, tPointi b)
{
    int i;
    for (i = 0; i < DIM; i++) a[i] = b[i];
}
```

5. PrintPoint. Escreve as coordenadas de um ponto.

```
void PrintPoint(tPointi p)
{
    int i;
    putchar('(');
    for (i = 0; i < DIM; i++) {
        printf("%d", p[i]);
        if (i ≠ DIM − 1) putchar(',');
    }
    putchar(')');
}
```

6. ReadPoints. Lê as coordenadas dos vértices de um polígono a partir da stdin, coloca os ponto em P, e retorna n, o número de vértices do polígono. É assumida que a entrada consiste de coordenadas dos vértices, separadas por brancos, e as coordenadas de um ponto por linha. O número de pontos não é parte da entrada.

```
int ReadPoints(tPolygoni P)
{
    int n = 0;
    printf("Polygon:\n");
    printf("␣␣i␣␣␣x␣␣␣y\n");
    while ((n < PMAX) ∧ (scanf("%d␣%d", &P[n][0], &P[n][1]) ≠ EOF)) {
        printf("%3d%4d%4d\n", n, P[n][0], P[n][1]);
        ++n;
    }
    if (n < PMAX) printf("n␣=␣%3d␣vertices␣read\n", n);
    else printf("Error␣in␣ReadPoints:␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣too␣many␣points;␣max␣is␣%d\n",
            PMAX);
    putchar('\n');
    return n;
}
```

7. main. Programa principal.

```
int main(void)
{
    int n;
    tPolygoni P;
    n = ReadPoints(P);
    Triangulate(n, P);
```

```
    return 0;
  }
```

9 de setembro de 1997, às 14:30

Algoritmo de Triangularização

Os códigos em `C` foram escritos por Joseph O'Rourke.

Este documento foi escrito em `CWEB`.