UAB

Estructures de dades dinàmiques

– Sessió 2 -

LP 2021-22

Exercici 4

Suposem que tenim creades les classes `Punt` I `Poligon` que hem utilitzat a la primera sessió de teoria, que serveixen per poder guardar un polígon a partir de la informació de cadascun dels seus vèrtexs:

```
class Punt
{
public:
   Punt();
   ~Punt();
   Punt(float x, float y): m_x(x), m_y(y) {};
   Punt(const Punt& pt);
   float getX() const { return m_x; }
   void setX(const float x) { m_x = x; }
   float getY() const { return m_y; }
   void setY(const float y) { m_y = y; }
private:
   float m_x;
   float m_y;
};
```

```
class Poligon
{
public:
   Poligon();
   ~Poligon();
   Poligon(int nCostats);

   int getNCostats() const { return m_nCostats; };
   bool afegeixVertex(const Punt &v);
   bool getVertex(int nVertex, Punt &v) const;
   float calculaPerimetre() const;
private:
   static const int MAX_COSTATS = 30;
   static const int MIN_COSTATS = 3;
   Punt m_vertexs[MAX_COSTATS];
   int m_nCostats;
   int m_nVertexs;
};
```

Exercici 4

Volem modificar la declaració de la classe `Poligon` per utilitzar un array dinàmic per guardar la informació dels vèrtexs i així poder guardar polígons amb un nº qualsevol de vèrtexs, sense límit:

1. Feu els canvis necessaris als atributs de la classe `Poligon` per substituir l'array estàtic original per un array dinàmic.
2. Modifiqueu els dos constructors i el destructor de la classe `Poligon` per adaptar el codi a la utilització de l'array dinàmic. La resta de mètodes de la classe `Poligon` no s'haurien de modificar.
3. Afegiu un constructor de còpia i un operador d'assignació a la classe `Poligon`.

Tingueu en compte:

- A Caronte trobareu la declaració i implementació original de les classes `Punt` i `Poligon` a modificar. També trobareu un programa principal que serveix per provar i validar la implementació de les classes. Mireu-vos el codi per entendre'l abans de fer les modificacions necessàries.
- En la implementació original, els constructors suposen que un polígon té un mínim de 3 costats i un màxim de 30. En la nova implementació el mínim s'ha de mantenir, però el límit màxim ha de desaparèixer.

Exercici 4 - Solució

```
class Poligon
{
public:
   Poligon();
   ~Poligon();
   Poligon(int nCostats);
   Poligon(const Poligon& p);
   Poligon& operator=(const Poligon& p);

   int getNCostats() const { return m_nCostats; };
   bool afegeixVertex(const Punt &v);
   bool getVertex(int nVertex, Punt &v) const;
   float calculaPerimetre() const;
private:
   static const int MIN_COSTATS = 3;
   Punt *m_vertexs;
   int m_nCostats;
   int m_nVertexs;
};
```

```
Poligon::Poligon()
{
    m_nCostats = MIN_COSTATS;
    m_vertexs = new Punt[m_nCostats];
    m_nVertexs = 0;
}

Poligon::~Poligon()
{
    delete [] m_vertexs;
}

Poligon::Poligon(int nCostats)
{
    if (nCostats > MIN_COSTATS)
        m_nCostats = nCostats;
    else
        m_nCostats = MIN_COSTATS;
    m_vertexs = new Punt[m_nCostats];
    m_nVertexs = 0;
}
```

Exercici 4 - Solució

```
Poligon::Poligon(const Poligon& p)
{
   m_nVertexs = p.m_nVertexs;
   m_nCostats = p.m_nCostats;
   m_vertexs = new Punt[m_nCostats];
   for (int i = 0; i < m_nVertexs; i++)
      m_vertexs[i] = p.m_vertexs[i];
}

Poligon& Poligon::operator=(const Poligon& p)
{
   if (this != &p)
   {
      if (m_vertexs != NULL)
          delete[] m_vertexs;
      m_nVertexs = p.m_nVertexs;
      m_nCostats = p.m_nCostats;
      m_vertexs = new Punt[m_nCostats];
      for (int i = 0; i < m_nVertexs; i++)
          m_vertexs[i] = p.m_vertexs[i];
   }
   return *this;
}
```

Exercici 7

Volem tornar a modificar la declaració de la classe `Poligon` de l'exercici anterior per guardar la informació dels vèrtexs utilitzant una estructura amb nodes dinàmics enllaçats:

1. Declareu i implementeu la classe `Node` que serà necessària per utilitzar una llista de punts amb nodes dinàmics enllaçats.
2. Feu els canvis necessaris als atributs de la classe `Poligon` per substituir l'array dinàmic de l'exercici anterior per una estructura dinàmica amb nodes enllaçats.
3. Modifiqueu tots els mètodes de la classe Poligon (inclosos el constructor de còpia i l'operador d'assignació) per adaptar el codi a la utilització de l'estructura amb nodes dinàmics.

- El mètode `afegeixVertex` serveix per afegir un nou vèrtex al polígon. Suposem que els vèrtexs es van afegint seqüencialment des del primer fins a l'últim. Si ja s'ha afegit el màxim de vèrtexs que indica l'atribut `m_nCostats`, es retorna `false`, en cas contrari, `true`.
- El mètode `getVertex` serveix per recuperar la informació del vèrtex que ocupa la posició dins del polígon que es passa com a paràmetre a `nVertex`, on `nVertex` ha de tenir un valor entre 1 i el nº de vèrtexs ja afegits al polígon. En cas contrari es retorna fals.

Exercici 7 - Solució

```
class Node
{
public:
   Node (const Punt& pt) { m_valor = pt;}
   void setValor(const Punt& pt) { m_valor = pt; }
   void setNext(Node* next) { m_next = next; }
   Punt getValor() const { return m_valor; }
   Node* getNext() const { return m_next; }
private:
   Punt m_valor;
   Node* m_next;
};
```

```
class Poligon
{
public:
   Poligon();
   ~Poligon();
   Poligon(int nCostats);
   Poligon(const Poligon& p);
   Poligon& operator=(const Poligon& p);

   int getNCostats() const { return m_nCostats; };
   bool afegeixVertex(const Punt& v);
   bool getVertex(int nVertex, Punt& v) const;
   float calculaPerimetre() const;
private:
   static const int MIN_COSTATS = 3;
   Node* m_vertexs;
   Node* m_ultimVertex;
   int m_nCostats;
   int m_nVertexs;
};
```

Exercici 7 - Solució

```
Poligon::Poligon()
{
    m_nCostats = MIN_COSTATS;
    m_nVertexs = 0;
    m_vertexs = NULL;
    m_ultimVertex = NULL;
}

Poligon::~Poligon()
{
    while (m_vertexs != NULL)
    {
        Node* aux;
        aux = m_vertexs;
        m_vertexs = m_vertexs->getNext();
        delete aux;
    }
}

Poligon::Poligon(int nCostats)
{
    if (nCostats > MIN_COSTATS)
        m_nCostats = nCostats;
    else
        m_nCostats = MIN_COSTATS;
    m_vertexs = NULL;
    m_ultimVertex = NULL;
    m_nVertexs = 0;
}
```

```
bool Poligon::afegeixVertex(const Punt& v)
{
    bool correcte = false;
    if (m_nVertexs < m_nCostats)
    {
        correcte = true;
        m_nVertexs++;
        Node* aux = new Node(v);
        if (m_vertexs == NULL)
            m_vertexs = aux;
        else
            m_ultimVertex->setNext(aux);
        m_ultimVertex = aux;
    }
    return correcte;
}
```

Exercici 7 - Solució

```
bool Poligon::getVertex(int nVertex, Punt& v) const
{
    bool correcte = false;
    if ((nVertex > 0) && (nVertex <= m_nVertexs))
    {
        Node* aux = m_vertexs;
        for (int i = 1; i < nVertex; i++)
            aux = aux->getNext();
        v = aux->getValor();
        correcte = true;
    }
    return correcte;
}
```

```
float Poligon::calculaPerimetre() const
{
    float perimetre = 0;
    float dx, dy;
    Node* aux = m_vertexs;
    for (int i = 0; i < (m_nCostats - 1); i++)
    {
        Node* seguent = aux->getNext();
        dx = aux->getValor().getX() - seguent->getValor().getX();
        dy = aux->getValor().getY() - seguent->getValor().getY();
        perimetre += sqrt(dx*dx + dy*dy);
        aux = seguent;
    }
    dx = aux->getValor().getX() - m_vertexs->getValor().getX();
    dy = aux->getValor().getY() - m_vertexs->getValor().getY();
    perimetre += sqrt(dx*dx + dy*dy);
    return perimetre;
}
```

Exercici 7 - Solució

```
Poligon::Poligon(const Poligon& p)
{
   m_nCostats = p.m_nCostats;
   m_nVertexs = 0;
   m_vertexs = NULL;
   m_ultimVertex = NULL;
   Node* aux = p.m_vertexs;
   while (aux != NULL)
   {
      afegeixVertex(aux->getValor());
      aux = aux->getNext();
   }
}
```

```
Poligon& Poligon::operator=(const Poligon& p)
{
   if (this != &p)
   {
      while (m_vertexs != NULL)
      {
         Node* aux;
         aux = m_vertexs;
         m_vertexs = m_vertexs->getNext();
         delete aux;
      }
      m_nCostats = p.m_nCostats;
      m_nVertexs = 0;
      Node* aux = p.m_vertexs;
      while (aux != NULL)
      {
         afegeixVertex(aux->getValor());
         aux = aux->getNext();
      }
   }
   return *this;
}
```

Exercici 9

Volem tornar a modificar la declaració de la classe `Poligon` de l'exercici anterior per utilitzar un objecte de la classe de la llibreria estàndard `std::forward_list` per guardar la informació dels vèrtexs:

1. Feu els canvis necessaris als atributs de la classe `Poligon` per utilitzar la classe `std::forward_list` per guardar l'estructura dinàmica amb nodes enllaçats.
2. Modifiqueu tots els mètodes de la classe `Poligon` per adaptar el codi a la utilització dels mètodes de la classe `std::forward_list`.

Exercici 9 - Solució

```
class Poligon
{
public:
   Poligon();
   ~Poligon();
   Poligon(int nCostats);
   Poligon(const Poligon& p);
   Poligon& operator=(const Poligon& p);

   int getNCostats() const { return m_nCostats; };
   bool afegeixVertex(const Punt& v);
   bool getVertex(int nVertex, Punt& v) const;
   float calculaPerimetre() const;
private:
   static const int MIN_COSTATS = 3;
   std::forward_list<Punt> m_vertexs;
   std::forward_list<Punt>::iterator m_ultimVertex;
   int m_nCostats;
   int m_nVertexs;
};
```

```
Poligon::Poligon()
{
   m_nCostats = MIN_COSTATS;
   m_nVertexs = 0;
   m_ultimVertex = m_vertexs.before_begin();}

Poligon::~Poligon()
{

}

Poligon::Poligon(int nCostats)
{
   if (nCostats > MIN_COSTATS)
      m_nCostats = nCostats;
   else
      m_nCostats = MIN_COSTATS;
   m_nVertexs = 0;
   m_ultimVertex = m_vertexs.before_begin();
}
```

Exercici 9 - Solució

```
bool Poligon::afegeixVertex(const Punt& v)
{
   bool correcte = false;
   if (m_nVertexs < m_nCostats)
   {
      correcte = true;
      m_ultimVertex = m_vertexs.insert_after(m_ultimVertex, v);
      m_nVertexs++;
   }
   return correcte;
}
```

```
bool Poligon::getVertex(int nVertex, Punt& v) const
{
   bool correcte = false;
   if ((nVertex > 0) && (nVertex <= m_nVertexs))
   {
      std::forward_list<Punt>::const_iterator aux = m_vertexs.begin();
      for (int i = 1; i < nVertex; i++)
         aux++;
      v = *aux;
      correcte = true;
   }
   return correcte;
}
```

Exercici 9 - Solució

```
float Poligon::calculaPerimetre() const
{
   float perimetre = 0;
   float dx, dy;
   std::forward_list<Punt>::const_iterator aux = m_vertexs.begin();
   std::forward_list<Punt>::const_iterator ant;
   for (int i = 0; i < (m_nCostats - 1); i++)
   {
      ant = aux;
      aux++;
      dx = ant->getX() - aux->getX();
      dy = ant->getY() - aux->getY();
      perimetre += sqrt(dx*dx + dy*dy);
   }
   dx = aux->getX() - m_vertexs.begin()->getX();
   dy = aux->getY() - m_vertexs.begin()->getY();
   perimetre += sqrt(dx*dx + dy*dy);
   return perimetre;
}
```

Exercici 9 - Solució

```
Poligon::Poligon(const Poligon& p)
{
   *this = p;
}

Poligon& Poligon::operator=(const Poligon& p)
{
   if (this != &p)
   {
      m_nCostats = p.m_nCostats;
      m_nVertexs = p.m_nVertexs;
      m_vertexs = p.m_vertexs;
      m_ultimVertex = m_vertexs.before_begin();
      std::forward_list<Punt>::iterator it;
      for (it = m_vertexs.begin(); it != m_vertexs.end(); it++)
          m_ultimVertex++;
   }
   return *this;
}
```

Informació disponible a Caronte

- Enllaç document “Guia d’estil de programació”
- Enllaç video “Instruccions debugger visual studio”
- Enllaç vídeos de repàs de teoria de MP
- Enllaç “Manual lliurament d’exercicis” pels problemes avaluables/no avaluables