ESTRUCTURAS LINEALES DINÁMICAS EN C++ BUILDER.

Autores: MsC. Marcos Antonio León Fonseca. mleonf@udg.co.cu
MsC. Noralys Muñiz Maldonado. nmunizm@udg.co.cu
MsC. Virgilio Herrera Rondón. vherrerah@udg.co.cu

Resumen:

En este artículo los autores muestran a través de un ejemplo, como se implementan las estructuras lineales dinámicas en el lenguaje de programación C++ Builder. El mismo es un resultado del trabajo metodológico desarrollado en la disciplina Lenguajes y Técnicas de Programación.

Summary:

In this article the authors show through an example, like take effect dynamic linear structures in the programming language C + + Builder. The same a result of the work methodological developed in discipline is Languages and programming Techniques.

Palabras claves: Estructuras dinámicas, listas enlazadas.

Introducción:

En el desarrollo de programas informáticos es común la utilización de datos de los cuales se desconoce su cantidad ya sea porque vengan como entrada en una cantidad no conocida de antemano o porque se generen en el programa en una forma no prevista.

Las estructuras de datos dinámicas le permiten a los programadores atender este tipo de situaciones ya que estas pueden ser fácil y eficientemente implementadas en los lenguajes de programación.

Este trabajo tiene como propósito mostrar como se implementan en el lenguaje de programación C++ Builder las estructuras lineales como casos particulares de las estructuras dinámicas.

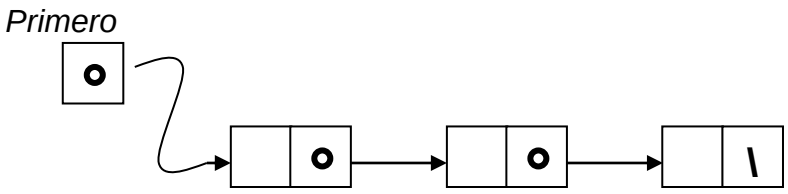
Desarrollo:

Las estructuras de datos que pueden variar en tamaño y en forma durante su tiempo de vida, reciben el nombre de estructuras dinámicas y se clasifican en lineales y no lineales.

Las estructuras lineales reciben ese nombre debido a que se puede hablar de un orden lineal de sus elementos; es decir, se puede decir que un elemento va antes o después que otro. Dentro de estas se encuentran las pilas, colas y listas enlazadas. Las pilas y colas son casos particulares de las listas enlazadas por lo que centraremos nuestra atención en estas últimas.

Una lista enlazada es una colección lineal de estructuras de un mismo tipo llamadas nodos, conectadas por enlaces de apuntador. Cada nodo se divide en dos partes: la primera contiene la información asociada a la estructura y la segunda la dirección del siguiente nodo de la lista.

El siguiente diagrama, muestra una lista enlazada compuesta por tres elementos:



Note como en esta representación gráfica, se incorpora una variable puntero llamada *Primero* que contiene la dirección del primer nodo de la lista y el campo puntero del último nodo contiene un valor especial, llamado valor nulo, representado por una diagonal, para señalar el final de la lista.

Para ilustrar como se implementan las listas enlazadas en C++ Builder, se analiza a continuación como se podría crear una lista para resolver la situación

que más adelante se describe, y una vez creada tener la posibilidad de visualizar, añadir o eliminar elementos de la misma.

En el Centro de Documentación e Información Pedagógica (CDIP) de la Universidad de Ciencias Pedagógicas “Blas Roca Calderío” de Manzanillo, se conoce de cada artículo publicado por el colectivo pedagógico de la institución: Título o tema de que trata el mismo, autor o fuente y la especialidad en que se enmarca la información lo que permite que los usuarios puedan preguntar por un artículo con un nombre determinado y se le informe si está o no en el CDIP y en caso de estar quien es el autor y la especialidad. Implemente un programa en C++ Builder para automatizar este proceso de forma que permita, no solo consultar información, sino agregar o eliminar artículos.

La creación de una lista enlazada se realiza mediante las estructuras autorreferenciadas. Una estructura autorreferenciada es aquella que contiene un campo de tipo puntero que apunta a una estructura del mismo tipo que la estructura que lo contiene. Por ejemplo, la definición:

```
struct Articulo {
 char *Titulo;
 char *Autor;
 char *Especialidad;
 struct Articulo *Siguiente;
};
```

contiene el campo *Siguiente* que es un puntero a la estructura de tipo *struct Articulo*.

En las listas enlazadas los datos se almacenan dinámicamente creando cada nodo según sea necesario. El final de una lista enlazada se señala definiendo a NULL el apuntador de enlace del último nodo de la misma.

En el diseño y la implementación de la clase *Publicacion*, aparecen los procedimientos *SetArticulo* y *DelArticulo* que permiten añadir y eliminar artículos a la lista enlazada y la función *GetAutor* que permite obtener el nombre del autor del artículo suministrado por el usuario.

```
class Publicacion {
 private:
  Articulo *Nuevo;
  Articulo *Primero;
  Articulo *Previo;
  Articulo *Actual;
 public:
  Publicacion ();
  void SetArticulo (char *, char *, char *);
  void DelArticulo (char *);
  char *GetAutor (char *);
  char *GetEspecialidad (char *);
};
```

```
Publicacion :: Publicacion () {
 Nuevo = NULL;
 Primero = NULL;
 Previo = NULL;
 Actual = NULL;
}

void Publicacion :: SetArticulo (char *titulo, char *autor, char *especialidad) {
 Nuevo = new Articulo;
 Nuevo->Titulo = new char [strlen (titulo) + 1];
 strcpy (Nuevo->Titulo, titulo);
 Nuevo->Autor = new char [strlen (autor) + 1];
 strcpy (Nuevo->Autor, autor);
 Nuevo->Especialidad = new char [strlen (especialidad) + 1];
 strcpy (Nuevo->Especialidad, especialidad);
 Nuevo->Siguiente = NULL;

 if (Primero == NULL) {
  Primero = Nuevo;
  Actual = Nuevo;
 }
 else {
  Previo = Actual;
  Actual = Nuevo;
  Previo->Siguiente = Actual;
 }
}
```

Se tiene acceso a los elementos de la lista enlazada a través del puntero al primer nodo de la lista y a partir de este a los restantes elementos mediante el apuntador de enlace que contiene cada nodo.

```
void Publicacion :: DelArticulo (char *titulo) {
 Articulo *Borrar = NULL;
 if (strcmp (Primero->Titulo, titulo) == 0){
  Borrar = Primero;
  Primero = Primero->Siguiente;
  delete Borrar;
 }
 else {
  Previo = Primero;
  Actual = Primero->Siguiente;
  while (Actual != NULL && strcmp (Actual->Titulo, titulo) != 0) {
   Previo = Actual;
   Actual = Actual->Siguiente;
  }
  if (Actual != NULL) {
   Borrar = Actual;
   Previo->Siguiente = Actual->Siguiente;
```

```
                    delete Borrar;
                   }
                  }
                 }
```

```
char *Publicacion :: GetAutor (char *titulo){
 Articulo *Temporal = NULL;
 Temporal = Primero;
 while (Temporal != NULL) {
  if (strcmp (Temporal->Titulo, titulo) == 0)
   return Temporal->Autor;
  Temporal = Temporal->Siguiente;
 }
 delete Temporal;
}
```

```
char *Publicacion :: GetEspecialidad (char *titulo){
 Articulo *Temporal = NULL;
 Temporal = Primero;
 while (Temporal != NULL) {
  if (strcmp (Temporal->Titulo, titulo) == 0)
   return Temporal->Especialidad;
  Temporal = Temporal->Siguiente;
 }
 delete Temporal;
}
```

Conclusiones:

El conocimiento de la implementación de las estructuras de datos dinámicas en el lenguaje de programación C++ Builder, permite el desarrollo de programas informáticos donde la cantidad de datos a procesar no es previsible.

Bibliografía:

Deitel, H. M. y Deitel., P. J. (2005). *Cómo Programar en C/C++.* Santiago de Cuba., PROGRAF.

Katrib Mora, M. (1986). *Lenguajes de programación y Técnicas de compilación.* Ciudad de la Habana., Editorial Pueblo y Educación.

Lipschutz, S. (1989). *Estructura de datos.* Ciudad de la Habana., Edición Revolucionaria.