Tutoría II verano Progra I

Tutora: Karla Verónica Quirós Delgado

Temas: Colecciones unidimensionales,

bidimensionales y lista simplemente enlazada.

Semana: 02-01-2023

Universidad Nacional de Costa Rica.

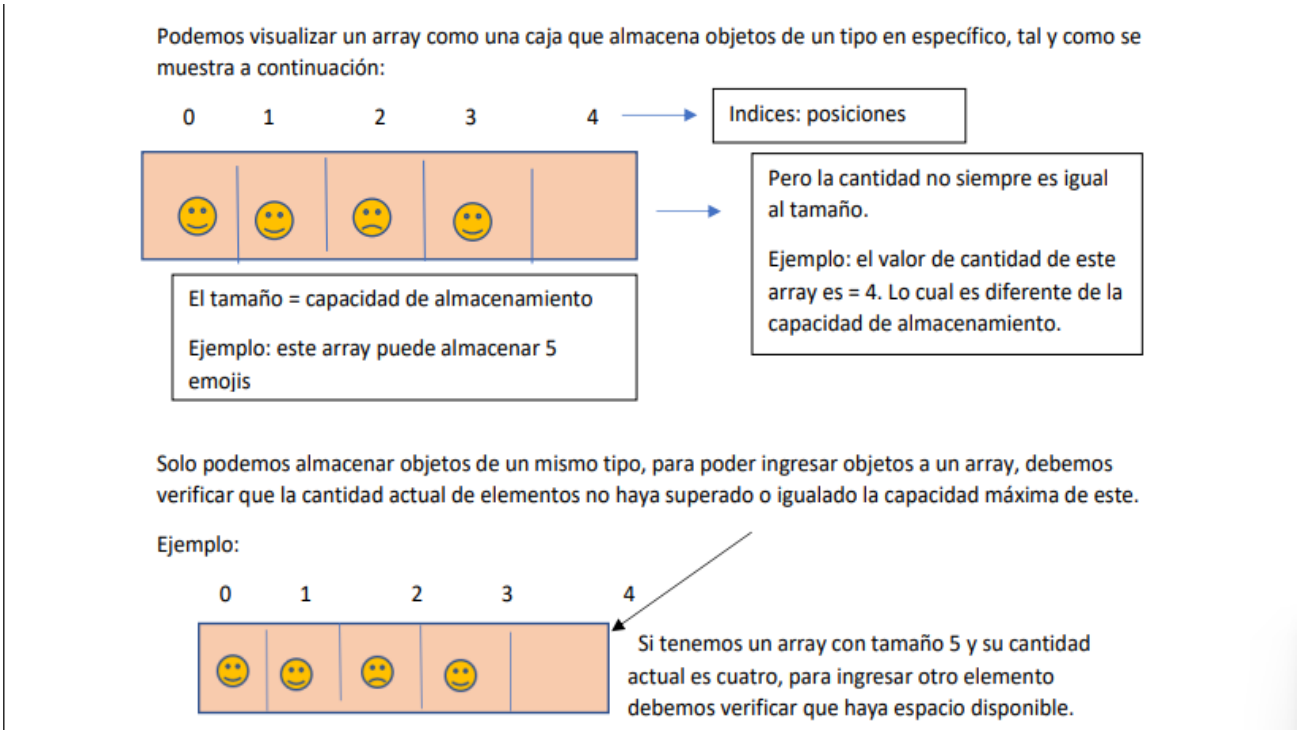
Colecciones Básicas:

Lo que se conoce como colecciones en el área de programación son estructuras en las que vamos a almacenar uno o más elementos, teniendo el acceso a estos y hacer operaciones tales como: ingresar, eliminar, modificar, sustituir y ordenar elementos que se encuentren en estas.

Unidimensionales:

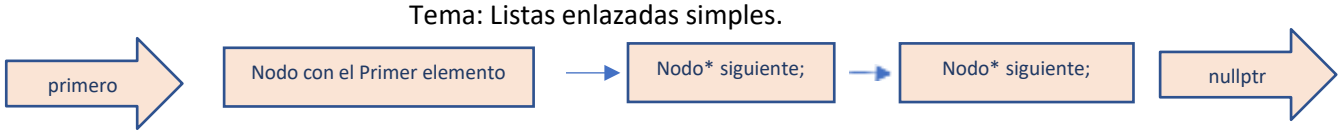
Un array es conocido como vector, estructura que almacena elementos de un mismo tipo. Tiene la capacidad de almacenar n elementos, su tamaño se define en su creación y esta caracterizado por algunos puntos primordiales:

1. Tamaño (definido al crear, capacidad máxima de almacenamiento).
2. Cantidad (número actual de elementos reales almacenados).
3. Se debe definir el tipo de datos que almacena: int vec[10] ejemplo de un vector de tamaño 10 y que almacena enteros .
4. El array siempre inicia desde el índice 0-n siendo n el tamaño definido de almacenamiento.
5. Se recorre utilizando un ciclo for que va a ir consultando posición por posición según los métodos necesarios en el array. Siempre se debe poner como condición que se recorra mientras no haya pasado el valor de cantidad de elementos válidos.



El array puede almacenar objetos dinámicos o automáticos, recordemos que los objetos automáticos liberan la memoria sin necesidad de que el programador lo realice. Sin embargo, los dinámicos recargan la responsabilidad de asignar y liberar la memoria de los objetos creados.

Tema: Listas enlazadas simples.



Para introducir este tema vamos a visualizar una lista como un o una serie de cajoncitos enlazados entre sí, que inicialmente tendrá solo un cajón vacío, luego se le irán asignando más cajones enlazados que van a contener elementos, tantos como se necesiten, las listas tienen características principales como:

- Estructura de programación que permite almacenar elementos de manera consecutiva.
- El espacio reservado no se hace de manera previa, se va asignando conforme se necesite, a diferencia de los vectores que se les asigna la memoria de forma previa al uso de este.
- El acceso a sus elementos no es directo, para acceder a un elemento habrá que recorrer sus nodos desde el primero hasta encontrarlo.
- Tiene inicialmente un puntero que será el primero en la lista (Nodo)el cual estará apuntando al primer Nodo de la lista, luego se irán enlazando nuevos Nodos.
- El ultimo Nodo de la lista estará apuntando a nullptr o NULL.
- Para cualquier operación de la lista se debe asignar lo que está viendo el primero (Nodo) a un nodo auxiliar, y con este se hará los recorridos necesarios, esto para no perder de vista al Nodo que apunta el primer cajoncito con un elemento.
- Los nodos son los cajoncitos donde almacenaremos los elementos, esta conformado por dos atributos propios: un puntero a Nodo* que será el nodo siguiente, además un puntero del tipo de dato a almacenar.

Ejemplo de una clase Nodo.h:

```
1.  Class Nodo {
2.  private:
3.  Tipo dato* dato;
4.  Nodo* siguiente;
5.  public:
6.    Nodo (); //por defecto
7.    Nodo (tipo de dato*, Nodo*); // parametrizado
8.    ~Nodo ();
9.     Nodo* getSiguiente();
10.    Void setSiguiente(Nodo*);
11.    Tipo dato* getDato();
12.    Void setDato(tipo dato*);
13. };
14.  // la construccion de un puntero a nodo=
15. Nodo* node= new Nodo (tipo de dato*, Nodo* siguiente);
16.
```

Ejemplo .cpp clase Nodo:

```
1.  Nodo(){
2.  dato = nullptr;
3.  siguiente = nullptr;
4.  }
5.  Nodo (tipo de dato* dato, Nodo* siguiente){
6.  this-> dato = dato;
7.  this->siguiente= siguiente;
8.  }
9.  ~Nodo () {
10. delete siguiente;
11. }
12. Nodo* getSiguiente(){
13. Return this->siguiente;
14. }
15. Tipo dato* getDato(){
16.  Return this->dato;
17. }
18. Void setSiguiente (Nodo* siguiente) {
19.  this->siguiente = siguiente;
20. }
21. Void setDato(tipo dato* dato){
22. This->dato = dato;
23. }
24.
```

Ejemplo básico de una clase Lista.h:

```
1.  #include"Nodo.h"
2.  Lista {
3.  private:
4.    Nodo* primero;
5.  Public:
6.           Lista();
7.          virtual ~ Lista ();
8.          void ingresaDato(Dato*);
9.          string toString();
10.         Nodo* getPrimero();
11. }
12.
```

Clase Lista.cpp

```
1.  Lista(){
2.  primero = nullptr;
    }
3.  ~Lista(){
4.    Nodo* p = nullptr; // nodo auxiliar
5.      while (primero != nullptr) {
6.          p = primero; // pondremos al uxiliar a ver lo que está apuntando el primero
7.          primero = primero->getSiguiente(); // el primero pasa a ver el siguiente
8.      delete p; // eliminamos el elemento al que apunte p en cada iteración9.
        }
10. }
11. Nodo* getPrimero(){
12. return this->primero;
13. }
14. Void ingresaDato(dato*){
15. Nodo* p = primero;
16.   if (primero == nullptr) { // en caso de que la lista este vacía
```

```
17.             primero = new Nodo(dato, primero);
18.   }
19.   else {// si la lista tiene al menos un elemento
20.             Nodo* nuevo = new Nodo(dato, nullptr);
21. //mientras el nodo auxiliar no apunte a nullptr aun no    hemos llegado al final
22. //si el que sigue de p ve a nullptr entonces podemos insertar el elemnto al final de la
    lista
23.             while (p->getSiguiente() != nullptr)
24.                       p = p->getSiguiente();
25.             p->setSiguiente(nuevo);
26.   }
27. }
28. String toString(){
29. Nodo* p = ppio;
30.     stringstream s;
31.         while (p != nullptr) {//mientras no apunte a una memoria nula
32.             s << p->getDato()->toString()<<endl;//imprima el elemento almacenado
33.             p = p->getSiguiente();// pase a ver el siguiente
34.         }
35.     return s.str();
36. }
37.
```

Nota:

¡EL ÉXITO LLEGA SIEMPRE A QUIENES NUNCA SE DAN POR VENCIDOS!

Atte:

Veronica Quiros Delgado.