Cuando necesitamos usar un arreglo de elementos, lo que normalmente se hace es declararlo, con un número fijo de elementos, por ejemplo:

```
int v [ 3 ];
```

Con esto hemos declarado un vector de enteros con 3 elementos, y accedemos a los respectivos valores mediante: v[0], v[1] y v[2]

Pero el inconveniente surge cuando no sabemos de antenamo, el número de elementos, pueden darse 2 casos:

1. Si declaramos por ejemplo un arreglo de 3 elementos pero usamos sólo un número menor, por ejemplo 2. La mayoría trabaja así y puede ser válido cuando trabajamos con arreglos de pocos elementos, pero si nuestro arreglo tiene unos 1000 elementos, obviamente estamos desperdiciando memoria.
2. El otro caso, es cuando el arreglo es de 3 elementos y necesitamos usar 4, otro problema.

La solución a esto es declarar un arreglo o puntero y asignar dinámicamente la memoria para los elementos adecuados, pero como hacemos esto??

Primero hacemos referencia a la librería malloc.h con la línea:

```
#include <malloc.h>
```

Y ahora si, fíjense en el siguiente código:

```
int elementos = 5 ;
int * v = NULL

v = ( int *) malloc
```

Dinámicamente asignamos memoria para 5 enteros, el valor de la variable elementos puede ser leída de teclado.

Ahora si tenemos una matriz nuestro código sería algo como:

```
int ** m = NUL
int filas = 5 ;
int columnas = 2 ;
int i ;

m = ( int **) malloc
for( i = 0 ;
```

Ahora, que sucede si luego queremos cambiar las dimensiones del vector o la matriz, pero sin perder la informacion que ya tenemos ?

Simplemente tenemos que reasignar la memoria, usando la función realloc.

Esta funcion reasigna la memoria que tenemos, es decir cambia las dimensiones de nuestra

variable, pero sin perder la información previa.

Para el caso de nuestro vector, supongamos que ahora queremos agregar 2 elementos más, es decir tenemos un total de 7 elementos, lo que haríamos es lo siguiente:

```
elementos = 7 ;
v = ( int *) realloc
```

Y para la matriz, digamos que queremos tener 100 filas y 50 columnas, lo que haremos es:

```
filas = 100 ;
columnas = 50 ;

m = ( int **) realloc
for( i = 0 ;
```

Con esto conseguimos manejar de forma optima la memoria y declaramos solo lo que vamos a usar.

Ahora, si tenemos por ejemplo una estructura como la siguiente:

```
    struct Estructura          {
                               string palabra              ;
                               int cant_docs               ;
};
```

Si no sabemos de antenamo cuantos elementos necesitamos, podemos vernos tentados a declarar un numero muy grande de elementos tal como:

```
    struct Estructura indice  [                              2000000                    ];
```

Es decir estamos declarando de inicio 2 millones de elementos, es probable que ni siquiera llegemos a usarlos todos, con los problema de memoria que puede traer esto.

Podemos usar la librería estándar de plantillas de C++ (STL), en concreto la llamada **<vector>** la cual permite tener un vector de tamaño variable, fíjense en el siguiente código:

```
          //   #include <vector>
#include <string>
#include <cstdlib>
#include <iostream>

using namespace std;

struct Estructura{
   string palabra;
   int cant_docs;
};

vector<Estructura>indice;

void Insert(const Estructura& e)
```

```
{
 indice.push_back(e);
}

void Delete(int i)
{
 if(i>=0)
  indice.erase(indice.begin()+i);
}

void Resize(int iMax)
{
 indice.resize(indice.size()+iMax);
}

void Show(int i)
{
 if(i>=0){
  cout << indice[ i ].palabra.c_str() << endl;
  cout << indice[ i ].cant_docs << endl;
 }
}

int main()
{
 Estructura e;

 e.palabra="palabra1";
 e.cant_docs=1;
 Insert(e);

 e.palabra="palabra2";
 e.cant_docs=2;
 Insert(e);

 cout << "\nSize: " << indice.size() << endl;
 cout << endl;
 for(int i=0; i< indice.size(); i++)
  Show(i);

 //Incrematamos el tamaño del vector
 Resize(10);
 cout << "\nSize: " << indice.size() << endl;

 system("pause");
 return 0;
```

}

Una vez más usamos sólo la cantidad de memoria adecuada, lo que hace a nuestro programa mucho más eficiente.

También se puede utilizar las plantillas que ofrece C++, como son <stack> -> pilas, <queue> -> colas o <list> -> listas

Para esto necesitaremos un compilador que soporte la librería estándar de plantillas como el **Dev-C++**

Para esta faq se han tomado los ejemplos que nuestro amigo **ajo** ha puesto en los Foros de este sitio.

Y listo eso es todo...