Asignación dinámica de memoria

En este documento vamos a ver como manejarnos con los punteros y la memoria, como hemos visto los punteros no son más que variables que contienen una dirección de memoria y sirven para señalar elementos dentro de esta.

La memoria está compuesta de celdas que contienen 1 byte de información, estás celdas están organizadas secuencialmente en direcciones desde la 0 hasta la cantidad que tengamos físicamente. Para acceder al contenido de una celda o byte debemos primero conocer la dirección que corresponde a dicho byte. Como ven la memoria es muy parecida a un vector de bytes.

En la memoria se almacenan los programas y los datos que usa, aunque en este documento nos concentraremos en los datos, recuerden siempre que por ahí también están los programas, por lo que andar modificando la memoria directamente sin saber que se está modificando puede acarrear modificar los programas. Nosotros vamos a hablar del uso de la memoria para las variables y estructuras en C/C++.

Asignación de memoria estática

Es la que hasta ahora hemos estado utilizando y es la que el compilador reserva antes de ejecutar el programa, por cada variable, arreglo o struct que declaramos el compilador ya sabe cuanta memoria necesita y es la que reserva, por ejemplo 4 bytes para los int, este tamaño no va a cambiar a lo largo del programa, lo cual para un int es algo perfecto, pero cuando declaramos, por ejemplo, un vector este permanecerá siempre del mismo tamaño, lo cual para algunas cosas ya no es tan perfecto, por ese motivo nos vemos obligados a sobredimensionar el vector al declararlo para asegurarnos que entre lo que venga, sin embargo hay casos que ni eso es suficiente.

Asignación dinámica de memoria

En esta forma desde el programa y en cualquier momento podemos pedir bloques de memoria para usarlos y luego devolverlos cuando ya no los necesitemos.
Para ello se utilizan un par de funciones,

en C:

malloc() para pedir memoria y free() para devolverla

en C++:

new para pedir memoria y delete para devolverla

Si bien ambos juegos de funciones cumplen con su cometido, nunca hay que mezclarlos, o sea si pedimos memoria con malloc hay que liberarla con free y lo mismo para el par new delete.

De ambos juegos de funciones vamos a usar new y delete tiene un uso más sencillo y servirá más adelante cuando se vean objetos.

Usando new y delete

new funciona dandole como parámetro la estructura que queremos guardar en memoria y new calcula cuanto espacio necesita, lo reserva y nos devuelve un puntero al comienzo del bloque reservado, por ejemplo para un int

int *p; //necesitamos un puntero donde almacenar la dirección del bloque

p=new int; //pedimos espacio para un entero

ahora podemos operar con el, por ejemplo asignarle el número 5 para eso usamos el operador *

*p=5;

podemos imprimir su contenido:

cout<<*p;

para liberar el espacio usamos el delete que hay que pasarle la dirección del bloque por lo que nunca hay que perderla, en nuestro caso:

delete p;

También podemos almacenar un arreglo, por ejemplo un vector de 10 elementos:

p=new int[10];

y acceder a sus elementos con notación de punteros, por ejemplo el 5to elemento:

*(p+4)=40;

o de la manera tradicional para el sexto:

p[5]=39; //recuerden la equivalencia entre punteros y arreglos

por supuesto, para liberar el espacio se usa:

delete p;

Con un struct

Creemos un struct con varios elementos:

```
struct ficha{
char nombre[50];
int edad;
float peso;
};
```

struct ficha f, *pf; //creamos una ficha estática (f) y un puntero ficha (pf)

pf=new struct ficha;

Ahora repasemos como accedíamos a los struct, para ello usábamos el operador punto (.), por ejemplo, para edad hacíamos:

f.edad=25;

o

cout<<f.edad;

para el resto:

f.peso=56.7

f.nombre[4]='A';

o para asignarle una cadena:

strcpy(f.nombre,”Juan”);

esto queda claro porque ya lo vimos en la unidad anterior, ahora ¿que pasa si tenemos un puntero como pf?, en ese caso en ves de usar el operador punto (.) usamos en operador flecha (→)

pf->edad=25;

o

cout<<pf->edad;

para el resto:

pf->peso=56.7

pf->nombre[4]='A';

o para asignarle una cadena:

strcpy(pf->nombre,”Juan”);

Es así de sencillo, por supuesto, para liberar la memoria usamos:

delete pf;

Lo úlltimo y más complicado sería un arreglo de struct

struct ficha f[10], *pf; //fijense que cuando es dinámico (pf) no necesito especificar cuantos elementos van a ser, eso lo voy a determinar cuando estoy ejecutando el programa, por ejemplo hagamos un pedido por teclado:

int n;

cin>>n;

pf=new struct ficha[n]; //generamos un arreglo de n elementos según sea el valor ingresado por teclado

ahora como accedemos, en forma estática a través del punto (.) como vimos en la unidad anterior

f[3].edad=25;
f[3].nombre[4]='A';

y en forma dinámica exactamente igual pero cambiando el punto por la flecha (->)

pf[3]->edad=25;
pf[3]->nombre[4]='A';

y para liberar obviamente

delete pf;

Aclaración: no es necesario en el delete que sea la misma variable puntero sino solo que contenga el mismo valor que devolvió new, por ejemplo, si creamos un puntero aux

struct ficha *aux;

y hacemos

aux=pf;

delete aux;

la memoria se liberará correctamente.

Ahora estamos listos para usar estructuras dinámicas.

Hagamos el concatenar pero con vectores dinámicos de tal manera que el CString resultante devuelva un vector que lo ajuste exactamente, en este caso devuelve el vector y los parámetros quedan invariables:

char * concatenar(const char *a, const char *b);

de tal manera que:

char *s;

s=concatenar(“Hola “,”Mundo”);

cout<<s; //debe imprimir Hola Mundo

el truco está en pedir memoria por el tamaño de la suma de ambas cadenas y luego copiarlas una tras de otra más o menos parecido a 5.1b, obviamente devolvemos el puntero que nos dio new

Y y que estamos duchos que tal que a s le insertemos un carater en la posición n

insertar(s,'A',3);

cout<<s; //debería imprimir HoAla Mundo

el prototipo sería

void insertar(char *s,char c,int n)

en este caso vale copiar parte del código de 5.1d

el truco, como sabemos que s vale exactamente lo suficiente para que entre Hola Mundo y ni una letra más, entonces necesitamos un nuevo sector de memoria un poco más grande, así que hay que pedir (new) el tamaño de s más 1, luego copiar s al nuevo sector con el caracter insertado, liberar s (delete) y asignar a s el nuevo sector

al final del main hacemos

```
delete s;
```

porque ya no usamos más a s