Autor: Lic. Kloster Daniel
Carrera: Técnico Universitario en Programación
Materia: Programación II
Tema: Asignación dinámica de memoria

Asignación dinámica de memoria

En el transcurso de las clases de las materias relacionadas con programación, hemos avanzado en el conocimiento y el uso de las estructuras de datos. Empezamos manejando variables simples, luego arrays (vectores y matrices), y punteros, y en las próximas clases llegaremos a crear, mediante el desarrollo de clases, variables registro que se ajustan a nuestras necesidades de procesamiento de información. Como veremos, de este tipo de variables también podemos declarar y utilizar arrays y punteros. La posibilidad de conocer variables más complejas nos permite abordar problemas más complejos también.

A pesar de todo esto, seguimos teniendo una limitación: todas las variables de memoria que conocemos son estáticas: debemos conocer previamente su tamaño para utilizarlas, ya que C/C++ nos exige declarar las variables antes de su uso. Si bien lo anterior no afecta a las variables simples, si resulta crítico para vectores y matrices.

Veamos el siguiente ejemplo:

1) Un comercio tiene un archivo (ventas.dat) con la información de las ventas con el siguiente formato:
- Nº de venta
- Fecha (dia)
- Importe unitario
- Cantidad vendida

La gerencia necesita hacer un análisis mensual de las ventas, por lo que solicita se desarrolle un programa para calcular e informar:

a) El importe recaudado por cada día del mes.
b) La cantidad de días sin ventas

El fin del ingreso de datos se indica con un número de venta igual a 0.

La resolución de ambos puntos sería, si suponemos que el mes de análisis es marzo (31 días), la siguiente:

Autor: Lic. Kloster Daniel
Carrera: Técnico Universitario en Programación
Materia: Programación II
Tema: Asignación dinámica de memoria

```
# include<iostream>


using namespace std;

void cargarDatos(float *v){
        int nVenta, dia, cantidad;
        float importe;
        cout<<"INGRESAR EL NUMERO DE VENTA ";
        cin>>nVenta;
        while(nVenta!=0){
            cout<<"INGRESAR EL DIA ";
            cin>>dia;
            cout<<"INGRESAR LA CANTIDAD ";
            cin>>cantidad;
            cout<<"INGRESAR EL IMPORTE UNITARIO ";
            cin>>importe;
            v[dia-1]+=importe*cantidad;
            cout<<"INGRESAR EL NUMERO DE VENTA ";
            cin>>nVenta;
        }
}

void puntoA(float *v, int tam){
    int i;
    for(i=0;i<tam;i++){
        cout<<"DIA "<<i+1<<" IMPORTE RECAUDADO "<<v[i]<<endl;
    }
}

int puntoB(float *v, int tam){
    int i, sinVentas=0;
    for(i=0;i<tam;i++){
        if(v[i]==0) sinVentas++;
    }
    return sinVentas;
}

int main(){
    float vImp[31]={0};
    cargarDatos(vImp);
    puntoA(vImp, 31);
    cout<<"CANTIDAD DE DIAS SIN VENTAS "<<puntoB(vImp, 31);
      cout<<endl;
      system("pause");
      return 0;
}
```

Como sabemos que son 31 días, en un vector de float de esa dimensión (vImp) acumulamos los importes parciales de cada venta. Para saber en qué posición del vector acumular los importes parciales, utilizamos el valor del día -1. Luego la función puntoA() muestra la recaudación diaria, y la otra calcula la cantidad de días sin ventas (son aquellas posiciones del vector con valor igual a 0).

Ahora ¿cómo resolver el problema si no sabemos el mes, y por lo tanto la cantidad de días?: no podemos declarar el vector ya que nos falta el tamaño, por lo cual el algoritmo sería mucho más complejo que el anteriormente visto.

Podríamos pedir el ingreso de la cantidad de días (o el mes a analizar) –lo cual nos resolvería el problema de no conocer la cantidad de días- pero igualmente seguiríamos sin la posibilidad de declarar el vector, ya que sólo podemos conocer la cantidad de días en el momento que se ejecute el programa (se dice en tiempo de ejecución), y no al momento de escribir el programa (se dice en tiempo de diseño). Para resolver esto existe un mecanismo que nos permite declarar dinámicamente las variables (recordemos que hasta ahora todas las variables son estáticas), que se denomina **asignación dinámica de memoria**.

Veamos cómo resolver el ejercicio mediante la asignación dinámica:

```
int main(){
    ///en un vector de 12 cargamos la cantidad de días de cada mes.
//Nos queda el problema del año biciesto
    int diasMeses[12]={31,28,31,30,31,30,31,31,30,31,30,31} ;
    int mes;
    cout<<"INGRESAR EL MES DE ANALISIS DE VENTAS ";
    cin>>mes;
    float *vImp;
    int dias=diasMeses[mes-1];//se obtiene la cantidad de días del mes
    vImp=new float[dias];///se pide memoria
    if(vImp==NULL){///se analiza si se pudo asignar la memoria
        cout<<"ERROR DE ASIGNACION DE MEMORIA";
        return -1;
    }
    ///desde esta línea se dispone del vector de float
    ponerCeroVector(vImp, dias);///función para poner en 0 el vector
//sobre el que se quiere acumular.

    ///el resto del programa se mantiene igual.
    cargarDatos(vImp);
    puntoA(vImp, dias);
    cout<<"CANTIDAD DE DIAS SIN VENTAS "<<puntoB(vImp, dias);
    cout<<endl;
    system("pause");
    delete []vImp;///se devuelve la memoria que se pidió
    return 0;
}
```

Las líneas de código agregadas o diferentes al programa anterior son:

```
int diasMeses[12]={31,28,31,30,31,30,31,31,30,31,30,31} ;/// se asigna en un vector la
cantidad de días de cada mes
float *vImp;// se declara un puntero a float
int mes;     // se declara una variable entera y se pide el ingreso del mes
vImp=new float[dias]// se pide memoria para un vector de float de una cantidad de dias
elementos
if(vImpArt==NULL){// se chequea que se haya podido asignar la memoria pedida
        cout<<"Error de asignación de memoria"<<endl;
        return -1;
        }
/////////
delete vImp; // se libera la memoria pedida
```

El resto de las líneas permanece igual.

Veamos en detalle cada línea:

float *vImp;
Como no se sabe la cantidad de elementos del vector no se lo puede declarar. Se declara un puntero a float, sobre el cual se pedirá luego memoria.

cout<<"INGRESAR EL MES DE ANALISIS DE VENTAS ";
cin>>mes;
int dias=diasMeses[mes-1]

se establece, a partir del mes, la cantidad de días para luego dimensionar el vector.

vImp=new float[dias];
Se pide memoria para un vector de dias elementos de tipo float.

El operador de C++ para asignar memoria es new. Su sintaxis es:

puntero_tipoX=new tipoX[cantidad_elementos]

Devuelve una dirección; se le debe indicar el tipo del vector y la cantidad de componentes entre corchetes.

Si la asignación pudo hacerse, la dirección que devuelve es donde empieza en la memoria el espacio solicitado; si no pudo hacerse la asignación, new devuelve NULL. Por esa razón es que

antes de continuar con el programa se analiza el valor del puntero sobre el que se solicitó memoria.

Volvamos a analizar la línea donde se pide memoria.

vImp=new float[dias];
como **new** devuelve una dirección se necesita un puntero para almacenarla

La última línea agregada fue

delete []vImp;

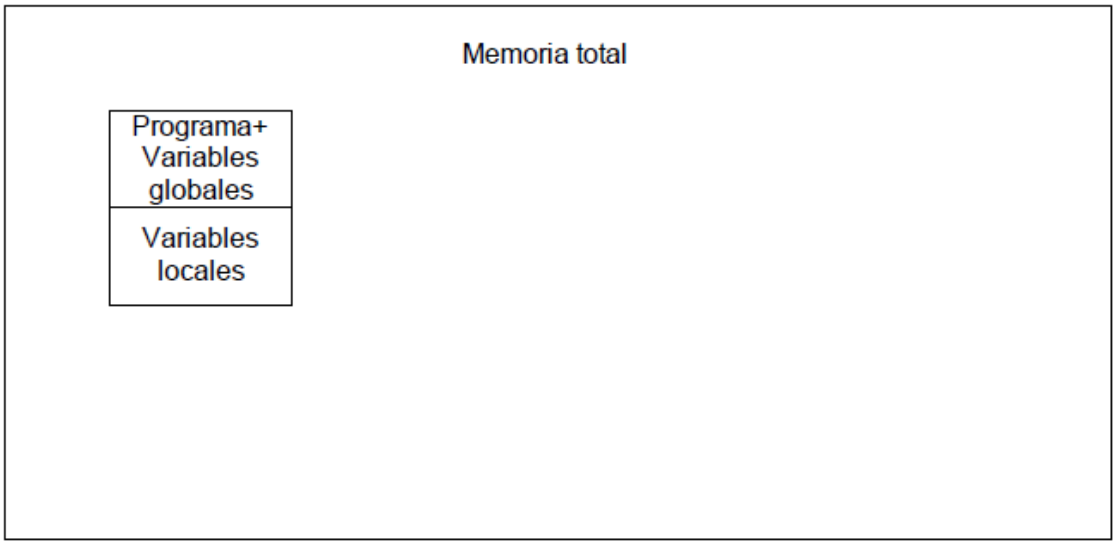
delete permite liberar la memoria pedida. Debe agregarse el puntero en el cual se hizo la asignación de memoria.

Autor: Lic. Kloster Daniel
Carrera: Técnico Universitario en Programación
Materia: Programación II
Tema: Asignación dinámica de memoria

Funcionamiento del mecanismo de asignación dinámica de memoria

Hemos visto cómo utilizar los operadores **new** para asignar dinámicamente memoria, y **delete** para liberar la memoria pedida. Veamos ahora como es su funcionamiento.

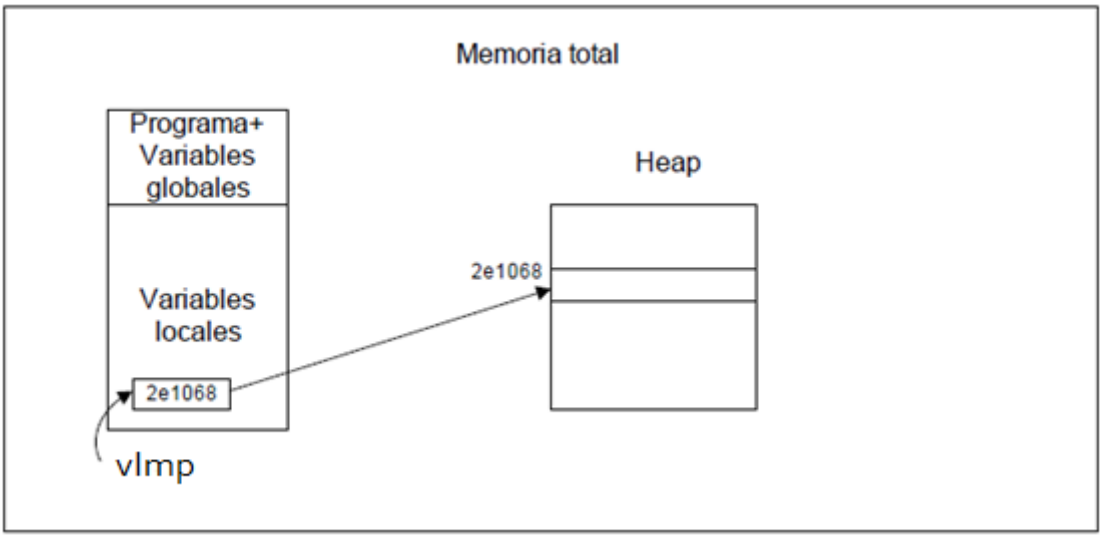
Al compilarse y enlazarse un programa, se reserva un espacio de memoria para el programa, las variables globales, y las variables locales a las funciones. El programa y las variables globales ocuparán un espacio fijo, equivalente a la suma de las necesidades del programa y la suma del tamaño de las variables globales declaradas; en otro espacio contiguo se reserva una pila para las variables locales, ya que no se utilizan todas al mismo tiempo. Podríamos representar lo anterior de la siguiente manera:



El programa sólo puede ejecutarse dentro del espacio de memoria que le ha sido asignado al cargarse por el sistema operativo. No se le permitirá acceder a direcciones de memoria fuera de esos límites.

Dentro del sistema de memoria, existe una zona denominada *heap* que puede ser utilizada de manera compartida por los programas en ejecución, usando asignación dinámica. Cuando los programas necesitan acceder a esas posiciones deben solicitarle al administrador de memoria del sistema operativo la cantidad de bytes requeridos, y en caso de disponer un bloque disponible de ese tamaño, el sistema operativo devolverá la dirección de memoria donde comienza ese bloque; caso contrario la respuesta será NULL. En C++ hacemos esta operación por medio de **new**; al terminar de usar el bloque de memoria, se la debe dejar libre para el caso que sea necesaria por otro programa. En C++ la liberación de memoria se hace por medio de **delete**.

Un esquema simplificado para el programa visto sería:



Dentro del área de memoria asignado al programa, se declara el puntero vImp. Cuando se ejecuta la línea:

vImp=**new** float[dias];

se le pide al sistema operativo un bloque de dias*sizeof (float) bytes contiguos para construir el vector; como hay disponibilidad en el *heap*, devuelve la dirección donde comienza el bloque (por ejemplo 2e1068), y esta dirección se asigna al puntero vImp.

Luego de la asignación, podemos utilizar vImp del mismo modo que utilizamos cualquier vector declarado de manera estática. Cuando no necesitamos más el vector, se libera la memoria solicitada con delete.