UNIVERSIDAD TECNOLÓGICA NACIONAL

Algoritmos y estructuras de datos

-2024-

DOCENTE: SANROMAN GABRIELA

TRABAJO PRÁCTICO

Localización [CAMPUS]					**Curso: K1201**							
Integrantes del equipo												
Legajo	**Apellido y Nombre**											
203.653-8.	Ezequiel García											
192.547-6	Jose Ezequiel Diaz											
218.373-0	Kamila Waigant											
213.672-7	Facundo Duhau											
213.680-6	Brisa Escobar											
Entrega / Revisión	**1**				**2**				**3**			
Fecha de entrega	28/07/2024											
Fecha de calificación												
Calificación	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**
Firma del Docente												

OBSERVACIONES:

Explicación del programa:

Se crearon 2 archivos .cpp:

- GenerarArchBinario.cpp
- Codigo.cpp

Con el primer archivo Hacemos un estruct de **Empresa** que tiene los datos:

- Código de empresa(**unsigned**)
- Nombre de empresa(**unsigned**)
- adherida(**bool**)

Con ese **struct** declaramos un vector con una dimensión máxima de 500 para poder cargar datos dentro de ese mismo vector y después poder escribirlos dentro de un archivo binario.

Utilizamos la función **cargarDatos()** para pedir datos por consola y poder guardarlos dentro del vector ya declarado. Además, esta función retornará un dato **unsigned** qué es la dimensión real que tendrá el vector donde se cargaron los datos, por el motivo de que no siempre sé cargará 500 empresas, sino que pueden ser menos.

```
unsigned cargandoDatos(Empresa vect[])
{
    unsigned cont = 0;
    unsigned auxCode;
    char valorAdherido[3];
    cout << "- Ingrese datos de empresa -" << endl;
    cout << "Dato de empresa[" << cont + 1 << "]" << endl;
    cout << "Codigo de empresa: ";
    cin >> auxCode;
    while (auxCode != 0)
    {
        vect[cont].codigo = auxCode;
        cout << "Nombre de Empresa: ";
        cin >> vect[cont].nombre;
        cout << "Esta adherisa al HotSale? ";
        cin >> valorAdherido;
        if (strcmp(valorAdherido, "si") == 0 || strcmp(valorAdherido, "SI") == 0)
        {
            vect[cont].adherido = true;
        }
```

```
        }
        else
        {
            vect[cont].adherido = false;
        }
        cont++;
        cout << "Dato de empresa[" << cont + 1 << "]" << endl;
        cout << "Codigo de empresa: ";
        cin >> auxCode;
    }
    return cont;
}
```

Luego se mostrarán los archivos de empresas ya cargadas por consola y también se mostrarán por consola las empresas adheridas al **Hotsale**.

```
void mostrandoVector(Empresa vect[], const int dim)
{
    cout << "Cantidad de empresas cargadas: " << dim << endl;
    for (int i = 0; i < dim; i++)
    {
        cout << "Codigo de empresa -> " << vect[i].codigo << " | Nombre de empresa -> " << vect[i].nombre << " | HotSale? -> " << vect[i].adherido << endl;
    }
}
```

```cpp
void mostrandoEmpresasAdheridas(Empresa vect[], const int dim)
{
    for (int i = 0; i < dim; i++)
    {
        if (vect[i].adherido)
        {
            cout << "Codigo de empresa -> " << vect[i].codigo << " | Nombre de empresa -> " << vect[i].nombre << endl;
        }
    }
}
```

Por último se crea una función **escribiendoArchivoDeEmpresas(),** que creará un archivo binario para escritura donde se colora los datos de todas las empresas.


```cpp
void mostrandoEmpresasAdheridas(Empresa vect[], const int dim)
{
    for (int i = 0; i < dim; i++)
    {
        if (vect[i].adherido)
        {
            cout << "Codigo de empresa -> " << vect[i].codigo << " | Nombre de empresa -> " << vect[i].nombre << endl;
        }
    }
}
```


Ingresando datos:

```
- Ingrese datos de empresa -
Dato de empresa[1]
Codigo de empresa: 11111
Nombre de Empresa: Macdonals
Esta adherisa al HotSale? si
Dato de empresa[2]
Codigo de empresa: 44444
Nombre de Empresa: Samsung
Esta adherisa al HotSale? si
Dato de empresa[3]
Codigo de empresa: 55555
Nombre de Empresa: LG
Esta adherisa al HotSale? si
Dato de empresa[4]
Codigo de empresa: 22222
Nombre de Empresa: Lenovo
Esta adherisa al HotSale? no
Dato de empresa[5]
Codigo de empresa: 0
```

Muestra de datos:

```
Cantidad de empresas cargadas: 4
Codigo de empresa -> 11111 | Nombre de empresa -> Macdonals | HotSale? -> 1
Codigo de empresa -> 44444 | Nombre de empresa -> Samsung | HotSale? -> 1
Codigo de empresa -> 55555 | Nombre de empresa -> LG | HotSale? -> 1
Codigo de empresa -> 22222 | Nombre de empresa -> Lenovo | HotSale? -> 0
```

Muestra de empresas adheridas:

```
 - Empresas adheridas al HotSale -
Codigo de empresa -> 11111 | Nombre de empresa -> Macdonals
Codigo de empresa -> 44444 | Nombre de empresa -> Samsung
Codigo de empresa -> 55555 | Nombre de empresa -> LG
```

Con el archivo.cpp:

Punto 1:

Definimos 2 vectores y un variable:

- vectorTotalImportePrueba(**unsigned**)
- vectorTotalCantVtaPrueba(**unsigned**)
- dimTotalImport(**unsigned**)

Creamos una función **totalEmpresasAdhe()** que recorre de forma anidada (for dentro de otro for) todo el vector de empresas adheridas sumando al vector **vectorTotalImportePrueba** el importe de las empresas por cada día que dura el Hot Sale.

Una vez terminado el recorrido se muestra por consola el código de empresa, total de empresas y total de importe

```
unsigned totalEmpresasAdhe(unsigned totalImport[], const int dimVta, Venta vtasEmpAdhe[], unsigned totalCantVta[])
{
    // hace los importes totales de cada empresa adherida
    unsigned posImporte = 0;
    for (int i = 0; i < dimVta; i += 3)
    {
        for (int j = i; j < (i + 3); j++)
        {
            totalImport[posImporte] += vtasEmpAdhe[j].importe;
            totalCantVta[posImporte] += vtasEmpAdhe[j].cantVentas;
        }
        posImporte++;
    }

    // muestra el importe total de cada empresa adherida por consola
    unsigned posAdhe = 0;
    for (unsigned h = 0; h < posImporte; h++)
    {
        cout << "Codigo de empresa -> " << vtasEmpAdhe[posAdhe].codigo << " | total cantidad de ventas -> " << totalCantVta[h] << " | Importe total -> " << totalImport[h] << endl;
        posAdhe += 3;
    }

    return posImporte;
}
```

la función retorna la dimensión que tendrá **vectorTotalImportePrueba y vectorTotalCantVtaPrueba** dentro de la variable **dimTotalImport.**

Muestra por consola:

```
- Total de las empresas Adheridas -
Codigo de empresa -> 11111 | total cantidad de ventas -> 21 | Importe total -> 5590
Codigo de empresa -> 44444 | total cantidad de ventas -> 5 | Importe total -> 850
Codigo de empresa -> 55555 | total cantidad de ventas -> 2 | Importe total -> 670
```

Punto 2:

Informar por cada empresa adherida, si corresponde, los días en que no realizaron ventas. (a cargo de Brisa Escobar)

```
void diassinventas(Venta ventas[], const int dimVentas)
{
    for (unsigned j = 0; j < dimVentas; j++)
    {
        if (ventas[j].cantVentas == 0)
        {
            cout << "Empresa: " << ventas[j].codigo << " | El dia: " << ventas[j].dia + 1 << " -> no tuvo ventas " << endl;
        }
    }
}
```

Se crea una función en la cual la llamamos diassinventas, y utiliza el struct ventas y el vector ventas y así mismo la dimensión de las ventas, en dicha función se recorre la dimensión de las ventas y si dentro del vector de ventas la cantidad de ventas en el día fue igual a cero, muestra por pantalla el código de la empresa, y el día que no tuvo ventas. Anteriormente se tenía en cuenta recorrer la empresa y los días del hotsale y usar un bool, pero para simplificar el código, se decidió solo recorrer la dimensión de las ventas y solo ver la cantidad de ventas realizadas por día.

Muestra por consola:

```
- Dias que no se realizaron ventas de empresas adheridas al HotSale -
Empresa: 11111 | El dia: 2 -> no tuvo ventas
Empresa: 55555 | El dia: 1 -> no tuvo ventas
Empresa: 55555 | El dia: 3 -> no tuvo ventas
```

Punto 3:

Se crea una función **menorImporteTotal()** que primero hace, buscar cuál es el menor importe de las empresas adheridas y mostrarla por pantalla la empresa, el código de la empresa e importe.

```
void menorImporteTotal(unsigned importTotal[], const int dimImporteTotal, int codigoEmpAdhe[], const int dimCodigoEmpAdhe)
{
    // enconetra cual es el menor importe total de las empresas adheridas
    unsigned menorImporte = importTotal[0];
    for (unsigned k = 0; k < dimImporteTotal; k++)
    {
        if (menorImporte > importTotal[k])
        {
            menorImporte = importTotal[k];
        }
    }

    for (unsigned i = 0; i < dimCodigoEmpAdhe; i++)
    {
        if (importTotal[i] == menorImporte)
        {
            cout << "La empresa que tuvo el menor importe total es -> " << codigoEmpAdhe[i] << " , con el importe de -> " << menorImporte << endl;
        }
    }
}
```

Muestra por consola:

```
La empresa que tubo el menor importe total es -> 55555 , con el importe de -> 670
```

Punto 4:

Informar cuál de los tres días hubo más ventas en total. (a cargo de Ezequiel Diaz)

```
void mayordiaventas(Venta vect[], int dim)
{
    int aux1 = 0, aux2 = 0, aux3 = 0, mayor = -1;
    int vecaux[3] = {0};

    for (int i = 0; i < dim; i++) // Recorro el vector con i=0 inicialmente, despues le sumo 3 y asi sucesivamente para la suma de todas las cant de ventas de todos los dias 1.
    {
        if (vect[i].dia == 0)
        {
            aux1 += vect[i].cantVentas;
        }
        else if (vect[i].dia == 1)
        {
            aux2 += vect[i].cantVentas;
        }
        else if (vect[i].dia == 2)
        {
            aux3 += vect[i].cantVentas;
        }
    }
```

```
    vecaux[0] = aux1;
    vecaux[1] = aux2;
    vecaux[2] = aux3;

    for (int j = 0; j < 3; j++) // Recorro el vector para buscar el mayor de cant de ventas.
    {
        if (vecaux[j] > mayor)
            mayor = vecaux[j];
    }

    cout << "Dia/s con mas ventas:" << endl;
    for (int k = 0; k < 3; k++) // Se muestra los dias con mayor ventas.
    {
        if (mayor == vecaux[k])
            cout << "Dia -> " << k + 1 << " | Cantidad de ventas -> " << vecaux[k] << endl;
    }
}
```

Se utilizó una función llamada mayordiaventas para informar en cuál de los tres días hubo más ventas de las empresas adheridas al hotsale. En esta función se recorre primero el vector struct ventas sumando la cantidad de ventas que hubo por cada dia (dia 1, dia 2, dia 3). Se utilizaron 3 condicionales para saber que venta pertenece a cada día. Los valores obtenidos se guardaron en un vector auxiliar de 3 espacios, en el cual el primer espacio se guardó la cantidad de ventas total que hubo en el primer dia, en el segundo espacio se guardó la cantidad de ventas total que hubo en el segundo dia y en el tercer espacio se guardó la cantidad de ventas total que hubo en el tercer dia. Después se recorre ese vector auxiliar para buscar el valor máximo. Una vez obtenido el valor máximo se recorre nuevamente el vector, usando un condicional se va comparando el contenido del vector auxiliar con valor máximo, si son iguales se muestra por pantalla el día y la cantidad de ventas total que hubo ese día.

Muestra por consola:

```
Dia/s con mas ventas:
Dia -> 3 | Cantidad de ventas -> 21
```

Punto 5:

Generar el archivo “ImporteEmpresas.dat”, con un registro por cada empresa adherida, ordenado por código de empresa. Cada registro deberá tener código y nombre de la empresa y el importe total recaudado por sus ventas y la cantidad de ellas. El archivo debe estar ordenado por código de empresa (a cargo de Facundo Duhau)

Para resolver este punto, era necesario utilizar los vectores y structs utilizados previamente, de un modo en el que todos los datos de las empresas adheridas puedan ser, primero, ordenados según el código vinculado a la empresa.

Para eso, lo que razonamos fue utilizar el método de burbujeo (Bubble Sort), de manera que si el código de una empresa es más grande que otro comparado, va a ser “empujado” al fondo del array.

Como todos los datos de las empresas se mueven juntos, esto nos facilita luego el ingreso de datos al archivo ImporteEmpresas.dat

```
void ordenamientoEmpresa(Empresa vectEmpAdhe[], const int dimEmpAdhe, unsigned posOrdenamiento[])
{
    Empresa temp;
    unsigned aux;
    unsigned auxPosicionOrd[dimEmpAdhe] = {0};

    for (int k = 0; k < dimEmpAdhe; k++)
    {
        auxPosicionOrd[k] = k;
    }

    for (int i = 0; i < dimEmpAdhe - 1; i++)
    {
        for (int j = 0; j < dimEmpAdhe - 1 - i; j++)
        {
            if (vectEmpAdhe[j].codigo > vectEmpAdhe[j + 1].codigo)
            {
                temp = vectEmpAdhe[j];
                vectEmpAdhe[j] = vectEmpAdhe[j + 1];
                vectEmpAdhe[j + 1] = temp;

                aux = auxPosicionOrd[j];
                auxPosicionOrd[j] = auxPosicionOrd[j + 1];
                auxPosicionOrd[j + 1] = aux;
            }
        }
    }
    posOrdenamiento = auxPosicionOrd;
}
```

Una vez ordenado todos los códigos de las empresas (y los datos de las empresas con ellos), hay que asignar a cada dato ingresado al archivo binario su espacio, el destino en el archivo declarado previamente, y escribirlo.

No hay que olvidar, que al intentar abrir el archivo, es importante que si “falla” por alguna razón, tiene que retornar información aclarando que falló, para así salir del programa, e intentar de nuevo.

```
void cargarArchivo(Empresa vectEmp[], unsigned importeTotal[], unsigned cantVtasTotal[], const int dimEmp, int codigoEmpAdheridas[], const int dimCodigoEmpAdhe, unsigned posOrdenamiento[])
{
    FILE *archivo = fopen("ImporteEmpresas.dat", "wb"); // Abre archivo binario.

    if (!archivo) // Si no lo puede abrir, devuelve error.
    {
        cout << "Error al intentar abrir el archivo"; // error archivo no encontrado...
        // exit(-1);
        return;
    }
    /* mostrarStruct(); */ // Copia el struct ordenado al archivo binario.
    int posImporteCantVts = 0;
    for (int i = 0; i < dimEmp; i++)
    {
        for (int j = 0; j < dimCodigoEmpAdhe; j++)
        {
            if (vectEmp[i].codigo == codigoEmpAdheridas[j])
            {
                fwrite(&vectEmp[i].codigo, sizeof(unsigned), 1, archivo);
                fwrite(&vectEmp[i].nombre, sizeof(char[30]), 1, archivo);
                fwrite(&cantVtasTotal[/* posImporteCantVts */ posOrdenamiento[i]], sizeof(unsigned), 1, archivo);
                fwrite(&importeTotal[/* posImporteCantVts */ posOrdenamiento[i]], sizeof(unsigned), 1, archivo);
                posImporteCantVts++;
            }
        }
    }
    fclose(archivo);
}
```

Punto 6:

Generar el archivo “SinAdhesion.dat”, con las ventas ingresadas de empresas que no estuvieron adheridas a la promoción. (a cargo de Kamila Waigant)

```
void generarArchivoSinAdhesion(Venta vectVentas[], const int dimVtas)
{
    FILE *archivo = fopen("SinAdhesion.dat", "wb");
    if (!archivo)
    {
        cout << "Error al abrir el archivo SinAdhesion.dat" << endl;
        return;
    }

    for (int i = 0; i < dimVtas; i++)
    {
        fwrite(&vectVentas[i], sizeof(Venta), 1, archivo);
    }
    fclose(archivo);
    cout << "Archivo SinAdhesion.dat generado exitosamente." << endl;
}
```

Se crea una función llamada “generarArchivoSinAdhesion”, cuyos parámetros son el struct “Venta” y la constante de dimensión de las ventas (dimVtas).

Esta función abre un archivo binario llamado “sinAdhesion.dat”. Si el archivo no se abre correctamente, muestra un mensaje de error y retorna.

Luego, con el ciclo for, recorre el array vectVentas y usa fwrite para escribir cada elemento en el archivo binario. Finalmente cierra el archivo luego de terminar de escribir los datos.