Tabla de contenidos

Resumen de la solución de la práctica.

1. Como solución a la práctica, **para almacenar los datos**, se ha optado usar las siguientes estructuras de datos:

    a. Para la **cinta numérica se ha empleado una cola**, mediante la implementación de celdas enlazadas, para ello se ha creado una **clase cola**. **Para representar las celdas de la cola**, en las cuales se almacenan los números, se ha creado una **clase** denominada **nodo_cola**.

    b. Para representar cada uno de los **montones se ha empleado una pila**, mediante la implementación de celdas enlazadas, para ello se ha creado una **clase pila**. **Para representar las celdas de la pila**, en las cuales se almacenan los números, se ha creado una **clase** denominada **nodo_pila**.

2. Para **generar los elementos del juego, números aleatorios, operadores aleatorios y realización de operaciones, se ha creado una clase** llamada **juego**.

3. El archivo **main.cpp** contiene la **función de inicio del programa**, esta función contiene los métodos necesarios, para **interactuar con el usuario, almacenar los datos introducidos por teclado y mostrar por pantalla los resultados**. Hace uso de todas las clases anteriores.

Índice de clases

Lista de clases

Lista de las clases, estructuras, uniones e interfaces con una breve descripción:

Indice de archivos

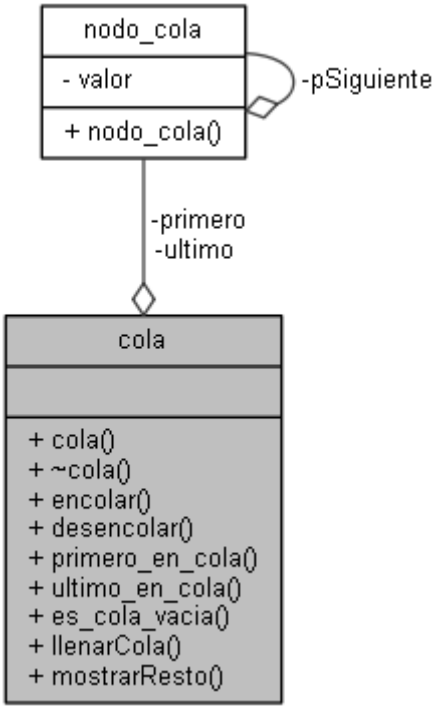
Lista de archivos

Lista de todos los archivos con descripciones breves:

Documentación de las clases

Referencia de la Clase cola

Estructura lineal para almacenar los números de la cinta.
`#include <cola.h>`
Diagrama de colaboración para cola:



Métodos públicos

- cola ()
  Constructor para crear una cola vacía.
- ~cola ()
  Destructor de la clase cola.
- void encolar (short elemento)
  Inserta un nuevo número en la cinta, cola.
- void desencolar ()
  Elimina el primer elemento de la cinta, cola.
- short primero_en_cola ()
  Devuelve el número que está almacenado en la primera celda de la cola, primero de la cinta.
- short ultimo_en_cola ()
  Devuelve el número que está almacenado en la última celda de la cola, último de la cinta.
- bool es_cola_vacia ()
  Comprueba si una cola, cinta, está vacía.
- void llenarCola ()

Función que llena la cola con un número n=5 de elementos.

- void mostrarResto ()
  Función que muestra por pantalla, todos los elementos de la cola menos el primero.

Atributos privados

- nodo_cola * primero
  Puntero de la clase nodo_cola para apuntar a la primera celda de la cola.
- nodo_cola * ultimo
  Puntero de la clase nodo_cola para apuntar a la última celda de la cola.

Descripción detallada

Estructura lineal para almacenar los números de la cinta.

Sigue una implementación de celdas enlazadas, cada celda es un objeto nodo_cola. La cola tiene una referencia a la primera celda y a la última:

```
-Si la cola está vacía, ambos punteros son "Null". primero=NULL, ultimo=NULL.

-Cada celda contiene un elemento y un puntero a la siguiente celda. (nodo_celda)
```

las inserciones de nuevos elementos solo se permiten en uno de los extremos de la cola, llamado último, y las consultas o eliminaciones solo se permiten en el opuesto, llamado primero. Sigue la estructura FIFO (First In, First Out)

Definición en la línea 25 del archivo cola.h.

Documentación del constructor y destructor

cola::cola ()

Constructor para crear una cola vacía.

No hay ningún nodo_cola creado, no hay celdas en la cola, ambos punteros primero y ultimo son "Null".

Definición en la línea 18 del archivo cola.cpp.

```
19 {            //Declaramos el valor inicial de las variables. Cola vacia.
20              primero=NULL;
21              ultimo=NULL;
22 }
```

cola::~cola ()

Destructor de la clase cola.

Elimina todas las celdas de la cola, recorriendo la cola desde la primera celda a la última, llama a la función desencolar(), hasta que el puntero primero es igual a NULL, ya no hay mas celdas

Ver también:

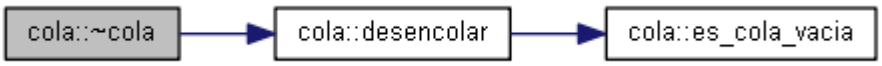
desencolar()

Definición en la línea 32 del archivo cola.cpp.

```
32              {
```

```
33      while (primero!=NULL){//Mientras primero apunte a una celda hay elementos.
34      desencolar();//quitamos el primer elemento.
35      }
36      //Despues de eliminar las celdas
37      //liberamos los punteros que ahora apuntan a null, tanto el primero como el
último.
38      delete primero;
39      delete ultimo;
40      cout<<"cola eliminada";
41
42  }
```

Gráfico de llamadas para esta función:



Documentación de las funciones miembro

void cola::desencolar ()

Elimina el primer elemento de la cinta, cola.

Elimina la primera celda de la cola, actualizando los punteros primero y último.

Variables:

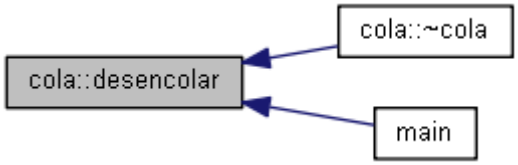
- p: puntero de la clase nodo_cola, apuntara al nodo que queremos eliminar

Definición en la línea 162 del archivo cola.cpp.

```
163 {
168       nodo cola*p=new nodo cola();
169
170       //Comprobamos la situación de la cola
171       if (es cola vacia()) //si esta vacía informa de ello.
172       {                    cout<<" La COLA está Vacia "<<endl;
173
174       }//fif
175       else//al estar llena
176        {
177                        p=primero;// p apunta al primer nodo
178                        primero=primero->pSiguiente;// El punetero primero se actualiza
apuntando al siguiente
179                        if(primero==NULL){//cola ya esta vacia
180                        ultimo=NULL;
181                        }
182
183        }//felse
184
185        //Se elimina ele elemento
186        p->pSiguiente=NULL; //por seguridad
187        delete (p);
188
189       return;
190 }
```

Gráfico de llamadas para esta función:

Gráfico de llamadas a esta función:



void cola::encolar (short *elemento*)

Inserta un nuevo número en la cinta, cola.

Inserta una nueva celda por el extremo último, un nuevo nodo_cola, con atributo valor=elemento y cuyo puntero a siguiente es igual a NULL

Parámetros:

elemento	número entero a insertar en la cola.

Variables:

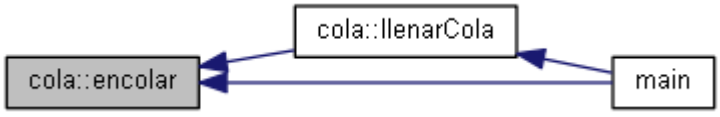
- p: puntero de la clase nodo_cola, el nodo al que apunta es la nueva celda a insertar.

Definición en la línea 71 del archivo cola.cpp.

```
72 {
77      nodo_cola*p=new nodo_cola();//puntero a nodo_cola. Nodo a insertar.
78      p->valor=elemento;//Al atributo valor, del nodo a insertar, le asignamos el
número almacenado en elemento.
79      p->pSiguiente=NULL;//El puntero del nodo a insertar toma el valor de NULL, pues
será el último nodo.
80
81      //Comprobara si la cola esta o no vacía, para colocar el nuevo nodo.
82      if (es_cola_vacia()) //en caso de no haber ningún elemento.
83      {
84      primero=p;//El puntero primero de la cola apuntara al nuevo nodo creado.
85      }
86      else // ya hay elementos en la cola.
87       {
88       ultimo->pSiguiente=p; //El puntero del nodo último apuntara al nuevo nodo
creado.
89       }
90
91     //una vez colocado el nodo actualizamos el puntero de la cola que apunta a la
última celda.
92       ultimo=p;//El puntero último de la cola apuntara al nuevo nodo creado e
introducido.
93
94 }
```

Gráfico de llamadas para esta función:

Gráfico de llamadas a esta función:



bool cola::es_cola_vacia ()

Comprueba si una cola, cinta, está vacía.

Comprueba si el puntero, primero, de la clase nodo_cola apunta a un nodo o a null. (primero==NULL) si es igual a null devuelve true en caso contrario devuelve false.

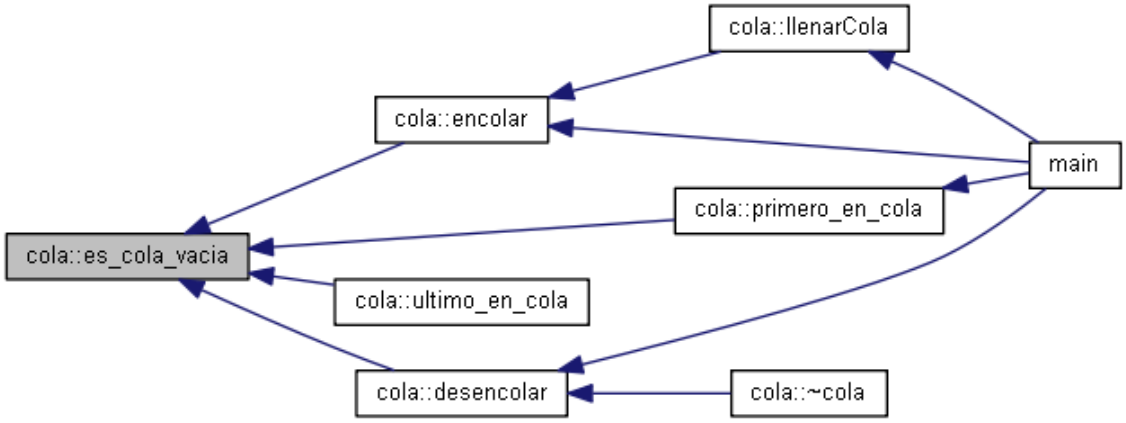
Valores devueltos:

true	la cola está vacía.
false	la cola tiene celdas, no está vacía.

Definición en la línea 51 del archivo cola.cpp.

```
52 {
53      if (primero==NULL)
54      {
55       return true;
56      }
57      else
58      {
59       return false;
60      }
61 }
```

Gráfico de llamadas a esta función:



void cola::llenarCola ()

Función que llena la cola con un número n=5 de elementos.

Desde 1 hasta n, genera un nuevo número aleatorio(entre 1-9) 'numeroAleatorio' y procede a encolarlo en la cola, llamando a la función encolar.

Ver también:

encolar(short numeroAleatorio)

Variables:

- n: Número de elementos a encolar, celdas que tendrá la cola ->5
- c: Variable que cuenta cada iteración del bucle hasta llegar a n
- numeroAleatorio: número entero que está entre 1 y 9 obtenido aleatoriamente.

Definición en la línea 200 del archivo cola.cpp.

```
200                     {
207     int n=5; //numero de elementos para encolar 5
208     int c=0; //Contador
209 for(c = 1; c <= n; c++){ //Desde 1 hasta llegar a 5
210 int numeroAleatorio=juego::obtenerNumero(); //Obtenemos un número entre 1 y 9
211 encolar(numeroAleatorio); //Se procede a encolar el número obtenido
212 }
```

```
213 }
```

Gráfico de llamadas para esta función:

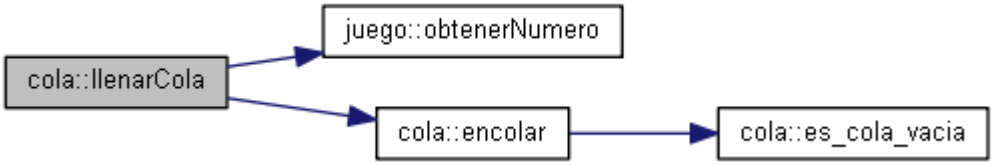


Gráfico de llamadas a esta función:

void cola::mostrarResto ()

Función que muestra por pantalla, todos los elementos de la cola menos el primero.

Recorre la cola desde el primer nodo hasta el último mediante un puntero auxiliar que va apuntando a los diferentes nodos de la cola, para cada nodo obtenemos el número que contiene su atributo valor y lo mostramos por pantalla, menos en el caso del primer nodo que no lo mostramos

Variables

- auxiliar: puntero a nodo_cola que va cambiando de valor, apuntando a los diferentes nodos de la cola, según avanza el recorrido

Definición en la línea 224 del archivo cola.cpp.

```
225     {
229         nodo_cola *auxiliar;
230 
231         auxiliar=primero; //inicializamos la variable auxiliar apuntando al 1°
elemento.
232 
233         while(auxiliar!=NULL) //mientras no apunte a NULL la cola continua. Hay
nodos.
234         {
235 
236             //En el caso de que auxiliar no apunte al primer nodo, imprimimos:
237             if (auxiliar!=primero){
238             //imprimimos los datos que nos dan los métodos del objeto al que apunta
el auxiliar.
239              cout<<auxiliar->valor<<" ";
240             }
241             //auxiliar pasa a apuntar al siguiente nodo, avanza el recorrido.
242             auxiliar=auxiliar->pSiguiente;
243             //if (auxiliar==NULL) break; //si auxiliar es igual a NULL ya no hay mas
nodos salimos
244         }
245 
246         delete auxiliar; //al salir de bucle liberamos el puntero.
247 
248     }
```

Gráfico de llamadas a esta función:

short cola::primero_en_cola ()

Devuelve el número que está almacenado en la primera celda de la cola, primero de la cinta.

Devuelve el número que está almacenado en el atributo valor, del primer nodo_cola, que pertenece a la cola. Mediante el puntero primero de la cola accedemos al valor del primer nodo.

Devuelve:

valor un entero comprendido entre el 1 y el 9.

Variables

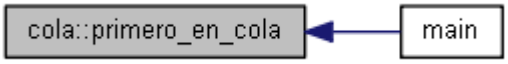
- e: Variable de tipo entero que almacena el valor del número primero

Definición en la línea 106 del archivo cola.cpp.

```
107 {
109     short e = 0;//Variable de tipo entero que almacena el valor del número primero.
110     try{
111
112      if (es_cola_vacia()){ //Si está vacía lanzamos la excepción
113      throw es_cola_vacia();
114
115      }//fif
116      else//Si está llena devuelve el numero del atributo valor del primer elemento
117       {
118                              e=primero->valor;//almacenamos en e
119       }//felse
120       }catch(bool){//Capturamos la excepción
121         cout<<"La cola está vacía";
122       }
123     return e;
124 }//fvoid
```

Gráfico de llamadas para esta función:

Gráfico de llamadas a esta función:



short cola::ultimo_en_cola ()

Devuelve el número que está almacenado en la última celda de la cola, último de la cinta.

Devuelve el número que está almacenado en el atributo valor, del último nodo_cola, que pertenece a la cola. Mediante el puntero ultimo de la cola accedemos al valor del último nodo.

Devuelve:

valor un entero comprendido entre el 1 y el 9.

Variables

- e: Variable de tipo entero que almacena el valor del número ultimo

Definición en la línea 134 del archivo cola.cpp.

```
135 {
138     short e = 0;//Variable de tipo entero que almacena el valor del número ultimo.
139     try{
140
```

```
141      if (es_cola_vacia()){ //Si está vacía lanzamos la excepción
142      throw es_cola_vacia();
143      }//fif
144      else//Si contiene celdas devolvemos el valor de la última.
145       {
146                           e=ultimo->valor;//almacenamos en e
147       }//felse
148
149   }catch(bool){//Capturamos la excepción
150     cout<<"La cola está vacía";
151   }
152     return e;
153 }//fvoid
```

Gráfico de llamadas para esta función:

Documentación de los datos miembro

cola::primero [private]

Puntero de la clase nodo_cola para apuntar a la primera celda de la cola.

Definición en la línea 36 del archivo cola.h.

cola::ultimo [private]

Puntero de la clase nodo_cola para apuntar a la última celda de la cola.

Definición en la línea 37 del archivo cola.h.

La documentación para esta clase fue generada a partir de los siguientes ficheros:

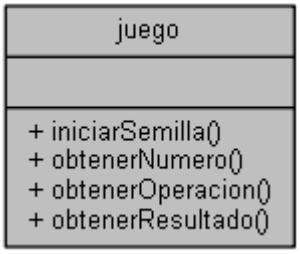
- archivos_h_cpp/cola.h
- archivos_h_cpp/cola.cpp

Referencia de la Clase juego

Clase que proporciona métodos estáticos, para el funcionamiento del juego.

`#include <juego.h>`

Diagrama de colaboración para juego:



Métodos públicos estáticos

- static void iniciarSemilla ()
  Método para iniciar, la generación del número aleatorio, mediante la hora del sistema.
- static short obtenerNumero ()
  Método para obtener un número entero aleatorio entre 1 y 9.
- static char obtenerOperacion ()
  Método para obtener un char, que simboliza la operación.
- static short obtenerResultado (char operacion, short numeroCinta, short cimaMonton)
  Función para realizar una operación, pasando dos enteros y el símblo del operador.

Descripción detallada

Clase que proporciona métodos estáticos, para el funcionamiento del juego.

Proporciona los métodos necesarios para obtener un numero aleatorio entre 1 y 9 generar una operación aleatoriamente, y operar para obtener un resultado.

Definición en la línea 17 del archivo juego.h.

Documentación de las funciones miembro

void juego::iniciarSemilla () `[static]`

Método para iniciar, la generación del número aleatorio, mediante la hora del sistema.

Usa el método srand() para iniciar la aleatoriedad usando el tiempo del sistema en milisegundos como semilla, para impedir que siempre salgan los mismos números.

Definición en la línea 16 del archivo juego.cpp.

```
16                          {
17      srand((unsigned)time(0));
18     //srand(time(NULL));
19 }
```

Gráfico de llamadas a esta función:

short juego::obtenerNumero () [static]

Método para obtener un número entero aleatorio entre 1 y 9.

Devuelve:

num Un número entero entre 1 y 9.

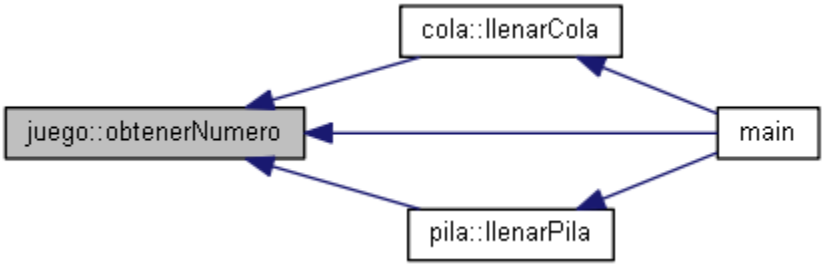
Variables

- num: almacena el número entero obtenido (1-9)

Definición en la línea 26 del archivo juego.cpp.

```
27 {
30     short num;
31     //rand();
32     num = 1 + rand() % (9);// rand() obtiene un número aleatorio en función de la
semilla
33     //Mediante la operador módulo obtenemos el resto de la operación (numero
aleatorio)/9
34     //Obligamos a que este dentro del intervalo entre 1 y 9 al sumarle 1
35
36     //Sleep(500);
37     return num;//Devolvemos el número.
38 }
```

Gráfico de llamadas a esta función:



char juego::obtenerOperacion () [static]

Método para obtener un char, que simboliza la operación.

Valores devueltos:

+	Simboliza suma.
-	Simboliza resta.
/	Simboliza división.
*	Simboliza multiplicación.

Variables:

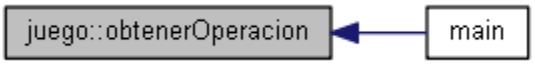
- operador: almacena el símbolo de la operación

- num: Almacena un número aleatorio del 1 al 4, en función de este num operador toma un valor si num=1 operador=*, si num=2 operador=/, si num=3 operador=+, si num=4 operador=-.

Definición en la línea 50 del archivo juego.cpp.

```
51 {
54     char operador;//almacena el símbolo de la operación
55     operador='p';//Inicializamos con p
56 
59     int num; //Almacena un número aleatorio del 1 al 4, en función de este operador
toma valor*/
60 
61         num = 1 + rand() % (4); //obtenemos el número aleatorio
62 
63         switch (num){// para cada número asignamos una operación.
64             case 1:
65                 operador='*';
66                 break;
67                 case 2:
68                 operador='/';
69                 break;
70                 case 3:
71                 operador='+';
72                 break;
73                 case 4:
74                 operador='-';
75                 break;
76         }
77 
78     return operador; //Devolvemos el caracter que simboliza la operación.
79     }
```

Gráfico de llamadas a esta función:



short juego::obtenerResultado (char *operacion*, short *numeroCinta*, short *cimaMonton*) `[static]`

Función para realizar una operación, pasando dos enteros y el símblo del operador.

En función del símbolo del operador aplica una operación matemática, a los dos números pasados,

Parámetros:

operacion	tipo char con el símbolo de la operación '+', '-', '*', o '/'.
numeroCinta	tipo short segundo termino de la operación.
cimaMonton	tipo short primer termino de la operación.

Devuelve:

resultado un entero que representa el resultado final de la operación
cimaMonton[operacion]numeroCinta

Variables

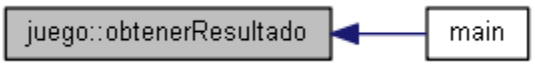
- resultado: almacena un entero con el resultado de la operación

Definición en la línea 91 del archivo juego.cpp.

```
91                                                                                {
94     short resultado=0;
95      switch(operacion){ //En función del char pasado
96 
97                 case('*'):{
98                                 //Realizamos la operación multiplicación
```

```
99                                  resultado=cimaMonton*numeroCinta;
100                                  break;
101                                  }
102                 case('/'):{
103                                  //Realizamos la operación división
104                                  resultado=cimaMonton/numeroCinta;
105                                  break;
106                                  }
107                 case('-'):{
108                                  //Realizamos la operación resta
109                                  resultado=cimaMonton-numeroCinta;
110                                  if (resultado<0){//Si es menor que cero
111                                     //Convertimos a valor absoluto.
112                                     resultado=-resultado;
113                                  }
114                                  break;
115                                  }
116                 case('+'):{
117                                  //Realizamos la operación suma
118                                  resultado=cimaMonton+numeroCinta;
119                                  break;
120                                  }
121             }
122     return resultado;//devolvemos el resultado.
123 }
```

Gráfico de llamadas a esta función:



La documentación para esta clase fue generada a partir de los siguientes ficheros:

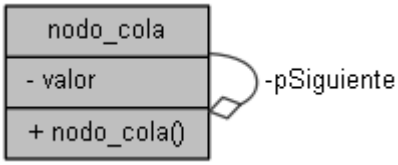
- archivos_h_cpp/juego.h
- archivos_h_cpp/juego.cpp

Referencia de la Clase nodo_cola

Representa una celda de la cola. Almacena cada uno de los números que componenen la cinta.

`#include <nodo_cola.h>`

Diagrama de colaboración para nodo_cola:



Métodos públicos

- nodo_cola ()
  Constructor de la clase nodo_cola, crea una celda que contiene un valor y un puntero.

Atributos privados

- short valor
  Contiene un número entero. El número almacenado en esa celda o nodo.
- nodo_cola * pSiguiente
  Puntero de la clase nodo_cola que a apunta al siguiente nodo_cola o celda de la cola.

Amigas

- class cola

Descripción detallada

Representa una celda de la cola. Almacena cada uno de los números que componenen la cinta.

Cada celda contiene un valor, un número entero y puntero a la siguiente celda de la cola. (Cola de celdas enlazadas)

Definición en la línea 10 del archivo nodo_cola.h.

Documentación del constructor y destructor

nodo_cola::nodo_cola ()

Constructor de la clase nodo_cola, crea una celda que contiene un valor y un puntero.

Crea un nuevo nodo aislado, cuyo puntero al siguiente apunta a null y contiene como valor el número 0.

Definición en la línea 14 del archivo nodo_cola.cpp.

```
14                      {
15              pSiguiente=NULL;//la celda no está enlazada con otra.
16              valor=0;//Asignamos el valor 0 como inicial de la celda
17
18 }
```

Documentación de las funciones relacionadas y clases amigas

friend class cola [friend]

Definición en la línea 16 del archivo nodo_cola.h.

Documentación de los datos miembro

nodo_cola::pSiguiente [private]

Puntero de la clase nodo_cola que a apunta al siguiente nodo_cola o celda de la cola.

Definición en la línea 21 del archivo nodo_cola.h.

nodo_cola::valor [private]

Contiene un número entero. El número almacenado en esa celda o nodo.

Definición en la línea 20 del archivo nodo_cola.h.

La documentación para esta clase fue generada a partir de los siguientes ficheros:

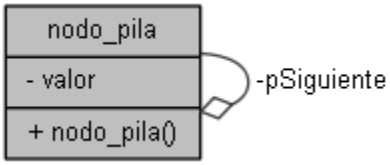
- archivos_h_cpp/nodo_cola.h
- archivos_h_cpp/nodo_cola.cpp

Referencia de la Clase nodo_pila

Representa una celda de la pila. Almacena cada uno de los números que componene un montón

`#include <nodo_pila.h>`

Diagrama de colaboración para nodo_pila:



Métodos públicos

- nodo_pila ()
  Constructor de la clase nodo_pila, crea una celda que contiene un valor y un puntero.

Atributos privados

- int valor
  Contiene un número entero. El número almacenado en esa celda o nodo.
- nodo_pila * pSiguiente
  Puntero de la clase nodo_pila que a apunta al siguiente nodo_pila o celda de la pila.

Amigas

- class pila

Descripción detallada

Representa una celda de la pila. Almacena cada uno de los números que componene un montón

Cada celda contiene un valor, un número entero y puntero a la celda que contiene lo que está debajo de ella (que a su vez es una pila).(Pila de celdas enlazadas)

Definición en la línea 10 del archivo nodo_pila.h.

Documentación del constructor y destructor

nodo_pila::nodo_pila ()

Constructor de la clase nodo_pila, crea una celda que contiene un valor y un puntero.

Crea un nuevo nodo aislado, cuyo puntero al siguiente apunta a null y contiene como valor el número 0.

Definición en la línea 14 del archivo nodo_pila.cpp.

```
14                      {//Constructor
15 pSiguiente=NULL;//No enlaza con otros nodos, no hay celdas enlazadas.
16 valor=0;//Se le asigna el número 0 al atributo valor, para inicializarlo.
17
18 }
```

Documentación de las funciones relacionadas y clases amigas

friend class pila [friend]

Definición en la línea 18 del archivo nodo_pila.h.

Documentación de los datos miembro

nodo_pila::pSiguiente [private]

Puntero de la clase nodo_pila que a apunta al siguiente nodo_pila o celda de la pila.

Definición en la línea 22 del archivo nodo_pila.h.

nodo_pila::valor [private]

Contiene un número entero. El número almacenado en esa celda o nodo.

Definición en la línea 21 del archivo nodo_pila.h.

La documentación para esta clase fue generada a partir de los siguientes ficheros:

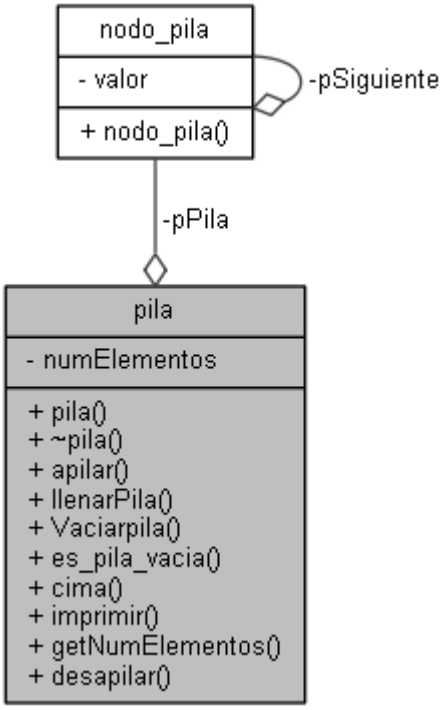
- archivos_h_cpp/nodo_pila.h
- archivos_h_cpp/nodo_pila.cpp

Referencia de la Clase pila

Estructura lineal para almacenar los números de los montones.

`#include <pila.h>`

Diagrama de colaboración para pila:



Métodos públicos

- pila ()
  Constructor para crear una pila vacía.
- ~pila ()
  Destructor de la clase pila.
- void apilar (short elemento)
  Función que pone un elemento, colocandolo como cima de la pila.
- void llenarPila ()
  Función que llena una pila con un número N (aleatorio) de elementos.
- void Vaciarpila ()
  Función que elimina todos los nodos creados. Eliminación de las celdas enlazadas.
- bool es_pila_vacia ()
  Función que comprueba si una pila esta vacía.
- short cima ()
  Función que devuelve el valor del elemento que se encuentra en la cima.
- void imprimir ()
  Función que imprime el contenido del atributo valor de los nodos almacenados en la pila.
- short getNumElementos ()
  Función que devuelve un entero con la longitud de la pila.

- void desapilar ()
  Función que elimina el elemento ,(nodo_pila), que es cima de la pila.

Atributos privados

- nodo_pila * pPila
  Puntero de la clase nodo_pila, apunta al nodo que es cima de la pila. Representa la pila.
- short numElementos
  Almacena el número de elementos que contiene la pila.

Descripción detallada

Estructura lineal para almacenar los números de los montones.

Sigue una implementación de celdas enlazadas, la pila se representa mediante un puntero a una celda, pPila. Cada celda es un objeto nodo_pila por lo que contiene un valor y un puntero a la siguiente celda. Estas celdas son colocadas una encima de la otra, la celda que entra en la pila, se coloca en la parte superior, siguiendo la estructura LIFO (Last In, First Out), la celda superior, es el último nodo_pila en entrar y se le llama cima, las inserciones, las consultas y las eliminaciones solo se permiten sobre la cima

Definición en la línea 18 del archivo pila.h.

Documentación del constructor y destructor

pila::pila ()

Constructor para crear una pila vacía.

No hay ningún nodo_pila creado, no hay celdas en la pila, por lo que el puntero a la cima, que reperesenta a la pila, toma el valor de null y numero de elementos que contiene la pila toma el valor de cero.

Definición en la línea 22 del archivo pila.cpp.

```
22           {
23             pPila=NULL;//inicialmente no hay nodo_pila que sea cima, el puntero toma
valor null
24             numElementos=0;//inicialmente no hay elementos en la pila, el contador
toma valor igual a cero
25 }
```

pila::~pila ()

Destructor de la clase pila.

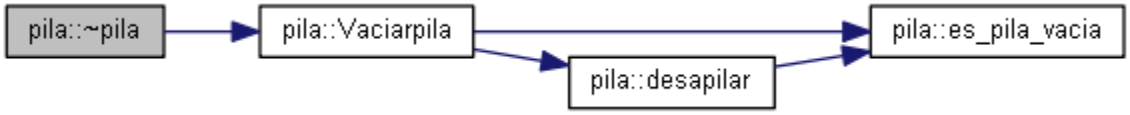
Elimina todos los nodos creados y libera pPila, puntero de la clase nodo_pila a la cima, que representa la cima.

Definición en la línea 33 del archivo pila.cpp.

```
33           {
34
35           Vaciarpila();//Función que elimina los nodos creados, partiendo del nodo
al que apunta el puntero pPila, cima.
36           delete pPila;//Una vez eliminados los nodos liberamos la pila.
37           cout<<"Pila eliminada";//informamos de la eliminación de la pila.
```

```
38 }
```

Gráfico de llamadas para esta función:



Documentación de las funciones miembro

void pila::apilar (short *elemento*)

Función que pone un elemento, colocandolo como cima de la pila.

Coloca un nuevo objeto nodo_pila en la cima de la pila, el puntero pSiguiente, de este nuevo nodo apunta al siguiente elemento de la pila y el puntero de la pila pPila apunta a este nuevo nodo que es la nueva cima.

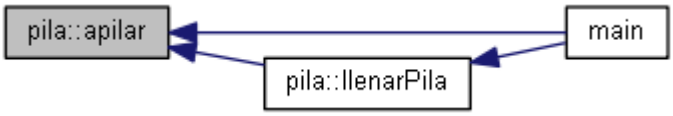
Parámetros:

elemento	número entero que queremos almacenar en la pila.

Definición en la línea 74 del archivo pila.cpp.

```
75 {
76 //proc apilar(p:pila, E e:elemento)
77 nodo pila*q=new nodo pila();//var q: enlace-pila; puntero a nodo pila, nuevo nodo a
introducir.
78 //reservar(q)
79 q->valor=elemento;//q^.valor<-e El atributo valor del nuevo nodo, toma el valor del
número que ha sido pasado.
80 q->pSiguiente=pPila;//q^.sig<-p El puntero, pSiguiente, del nuevo nodo, apunta al
nodo que ahora es cima, toma el valor de pPila.
81 pPila=q;//p<-q; el puntero de la pila, pPila, apunta al nuevo nodo introducido,
quedando así como cima.
82 numElementos++;// aumentamos en uno el valor del atributo numElementos que determina
el número de elementos que tiene la pila
83 //fproc
84 }
```

Gráfico de llamadas a esta función:



short pila::cima ()

Función que devuelve el valor del elemento que se encuentra en la cima.

Devolvemos el valor del atributo valor del nodo_pila al que apunta el puntero pPila, si es una pila vacía este puntero será null, (pPila==Null), en este caso lanzará una excepción.

Devuelve:

e El valor del número entero que se encuentra en la cima de la pila.

Definición en la línea 135 del archivo pila.cpp.

```
135                 {
136     short e=0;//Variable de tipo entero que almacena el valor del número que está en
la cima
```

```
137 try{
138 
139 if (es pila vacia()){
140 throw es pila vacia();//Si es una pila vacia lanzamos la excepción
141 }else{
142 e=pPila->valor;//Asignamos el valor a e, del atributo valor del nodo_pila, al que
apunta el puntero de la pila.
143 }
144 }   catch(bool){//Capturamos la excepción lanzada si la pila está vacía.
145     cout<<"La pila esta vacia";//Si hay excepción lo mostramos.
146 }
147 return e;//Devolvemos el atributo valor del nodo_pila que es cima de la pila.
148 }
```

Gráfico de llamadas para esta función:

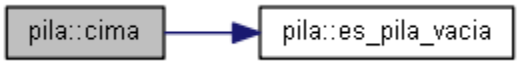


Gráfico de llamadas a esta función:

void pila::desapilar ()

Función que elimina el elemento ,(nodo_pila), que es cima de la pila.

Elimina el nodo_pila que es cima, actulizando el valor del puntero de la pila, pPila, que tomará el valor del siguiente nodo, adyacente al que que va a ser eliminado.

Variables

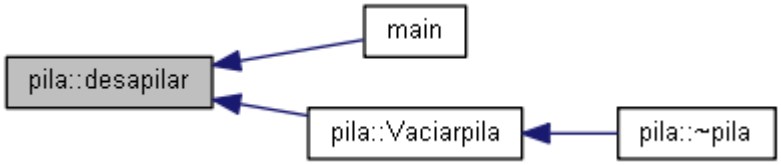
- Variable auxiliar q: un puntero a nodo_pila, que apuntará al nodo_pila a eliminar, la cima,

Definición en la línea 168 del archivo pila.cpp.

```
168                     {
169 //proc desapilar(p:pila)
172 nodo pila*q=new nodo pila(); //var q: enlace-pila puntero a nodo pila.
173 
174 if (es pila vacia()){
175 cout<<"La pila está vacía"  ;//si es_pila_vacia(p) entonces error(Pila vacia)
176 }else{ //si no
177 q=pPila; //q<-p q apunta al nodo_pila, cima, que es el nodo a eliminar.
178  //Actualizamos el puntero de la pila, una nueva cima.
179 pPila=pPila->pSiguiente; //p<-p^.sig Apunta al nodo que está a continuación del nodo
que se va a eliminar.
180 q->pSiguiente=NULL; //q^.sig<-nil {por seguridad} El nodo a eliminar no tiene que
tener un puntero a ningún otro nodo
181 delete(q); //liberar(q)
182 numElementos--; //Descontamos en uno el numero de elementos que tiene la pila.
183 } //fsi
184  //fproc
185 }
```

Gráfico de llamadas para esta función:

Gráfico de llamadas a esta función:



bool pila::es_pila_vacia ()

Función que comprueba si una pila esta vacía.

En función del valor de puntero pPila, determina si está vacía, si tiene valor null es una pila vacía.

Valores devueltos:

true	si la pila está vacia.
false	si la pila no está vacía

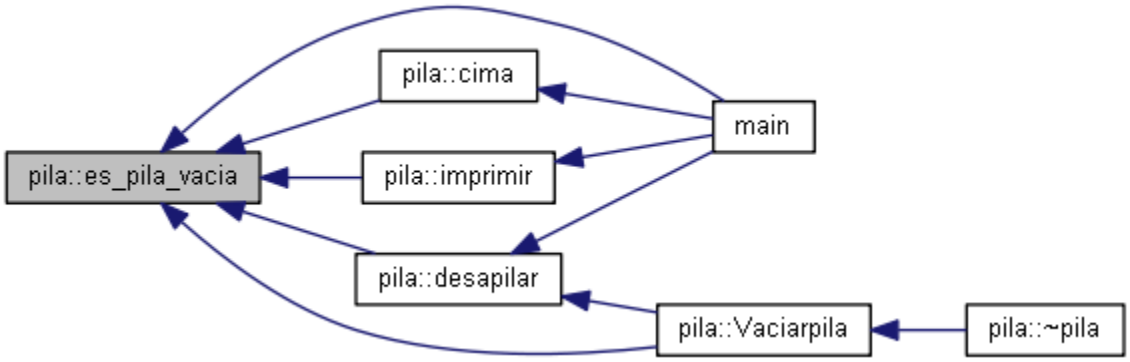
Variables:

- b: variable booleana que alamacena el valor del resultado (pPila==NULL)

Definición en la línea 117 del archivo pila.cpp.

```
117                         {
121 bool b;//almacena valor true o false
122 b=(pPila==NULL);//en función de si es true o false (pPila==NULL)
123 return b;//devolvemos valor de b
124 }
```

Gráfico de llamadas a esta función:



short pila::getNumElementos ()

Función que devuelve un entero con la longitud de la pila.

Devuelve un entero que representa el número de elementos que están almacenados en la pila.

Valores devueltos:

numElementos=0	La pila no tiene elementos.
0<numElementos <=9	Numero entero entre 0 y 9, que representa el número total de celdas que tiene la pila.

Definición en la línea 158 del archivo pila.cpp.

```
158                             {
159 return numElementos;
```

```
160 }
```

Gráfico de llamadas a esta función:

void pila::imprimir ()

Función que imprime el contenido del atributo valor de los nodos almacenados en la pila.

Muestra por pantalla el numero que está contenido en el atributo valor de cada objeto nodo_pila que se almacena en la pila. Recorre la pila desde nodo más superior, cima, hasta el nodo mas inferior fondo.

Se mostrará de la siguiente manera fondo,...nodos,... cima. Sentido de izquierda a derecha, empezando desde el primer elemento que entro en la pila, fondo, y como último elemento más a la derecha el nodo cima, último elemento en entrar en la pila.

Variables:

- contenido: una cadena que va almacenando el contenido de la pila según avanza el recorrido por los nodos.

ss: objeto stringstream usamos para almacenar el entero contenido en el atributo valor, y posteriormente convertirlo en una cadena string

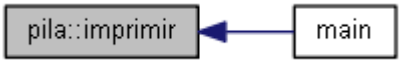
- q: variable auxiliar puntero a nodo_pila la usamos para ir recorriendo nodo a nodo la pila desde la cima
- str: objeto string que almacena el valor del entero, formateado a cadena, que se obtiene en cada nodo del recorrido

Definición en la línea 198 del archivo pila.cpp.

```
198                     {
201 string contenido="";//No hay contenido
204 std::stringstream ss;//Almacena los enteros que va recibiendo.
205
206 if (es pila vacia()){//Si es pila vacía no hay datos.
207 return;
208 }
210 nodo pila*q=new nodo pila();//var q: enlace-pila
211 q=pPila;//q apunta inicialmente al nodo que es la cima
212 while (q!=NULL){    //Mientras existan nodos
213
214 ss<<q->valor; //Almacenamos, el número contenido en el atributo valor del nodo, en
el objeto ss (stream)
216 std::string str = ss.str();//formateamos el contenido del stream a tipo cadena y lo
almacenamos en el objeto string str
217 contenido=str+", "+contenido;//El contenido anterior lo añadimos por la derecha.
218 ss.str("");//Vaciamos el stream
219
220       q=q->pSiguiente;// Ahora q apunta al siguiente nodo de la pila
221
222 }
223       delete(q);//liberar(q) al finalizar el recorrido.
224
225
226 cout<<contenido;//Mostramos el contenido por pantalla.
227 }
```

Gráfico de llamadas para esta función:

Gráfico de llamadas a esta función:



void pila::llenarPila ()

Función que llena una pila con un número N (aleatorio) de elementos.

Obtiene un número aleatoriao N entre 1 y 9 que determina el número de elementos que tendrá la pila. Desde 1 hasta N, genera un nuevo número aleatorio(entre 1-9) 'numeroAleatorio' apilando este número.

- n: Almacena el numero de elementos que ha de contener la pila.
- c: Contador para iterar el número n de elementos a introducir.
- numAleatorio: Es un entero que almacena el número que ha de ser apilado

Definición en la línea 94 del archivo pila.cpp.

```
95 {
97 int n=juego::obtenerNumero();//Obtenemos un numero entre 1-9 y lo asignamos a n.
99 int c=0; //Contador para el número N de elementos a introducir.
100 for(c = 1; c <= n; c++){//desde 1 hasta el numero de elementos a apilar
102 int numeroAleatorio=juego::obtenerNumero();//Obtenemos un número del 1 al 9
103 apilar(numeroAleatorio);//apilamos el número.
104 }
105
106 }
```

Gráfico de llamadas para esta función:

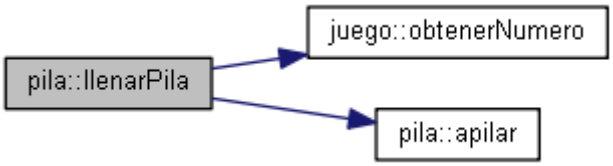


Gráfico de llamadas a esta función:

void pila::Vaciarpila ()

Función que elimina todos los nodos creados. Eliminación de las celdas enlazadas.

Elimina todos los nodos creados y libera el puntero pPila, nodo_pila a la cima. Llama a llama a la función desapilar hasta que no quede ninguna celda

Ver también:

desapilar()

Definición en la línea 47 del archivo pila.cpp.

```
47                          {
```

```
48
49  //nodo_pila*q=new nodo_pila();//var q: enlace-pila
50 if (es_pila_vacia()){
51 cout<<"La pila está vacía"  ;//si es_pila_vacia(p) entonces error(Pila vacia)
52 }else{//si no
53
54 while (pPila!=NULL){
55
56       desapilar();
57 }
58 }//fsi
59
60
61 }
```

Gráfico de llamadas para esta función:

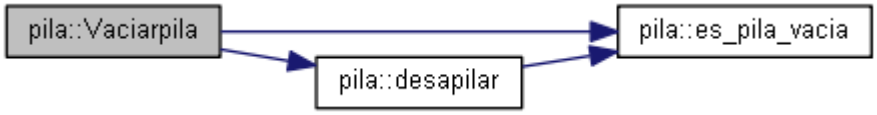


Gráfico de llamadas a esta función:

Documentación de los datos miembro

pila::numElementos `[private]`

Almacena el número de elementos que contiene la pila.

Definición en la línea 31 del archivo pila.h.

pila::pPila `[private]`

Puntero de la clase nodo_pila, apunta al nodo que es cima de la pila. Representa la pila.

Definición en la línea 30 del archivo pila.h.

La documentación para esta clase fue generada a partir de los siguientes ficheros:

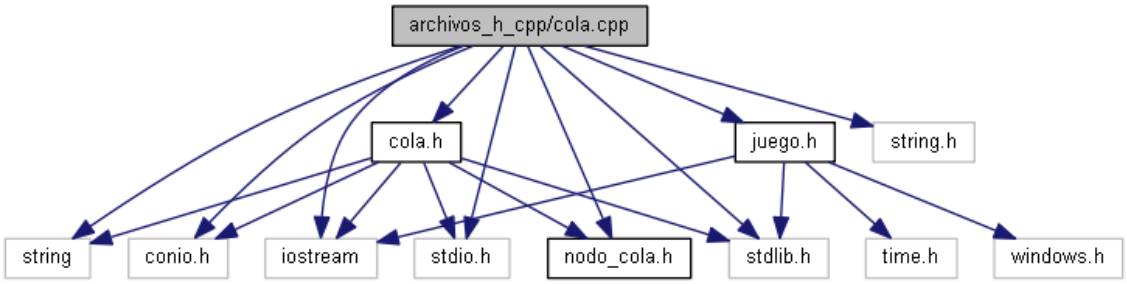
- archivos_h_cpp/pila.h
- archivos_h_cpp/pila.cpp

Documentación de archivos

Referencia del Archivo archivos_h_cpp/cola.cpp

```
#include <iostream>
#include <string>
#include <conio.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "cola.h"
#include "nodo_cola.h"
#include "juego.h"
```

Dependencia gráfica adjunta para cola.cpp:



Referencia del Archivo archivos_h_cpp/cola.h

```
#include <iostream>
#include <string>
#include <conio.h>
#include <stdlib.h>
#include <stdio.h>
#include "nodo_cola.h"
```

Dependencia gráfica adjunta para cola.h:

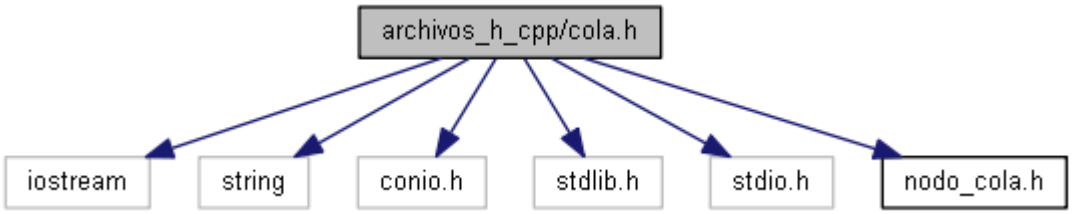
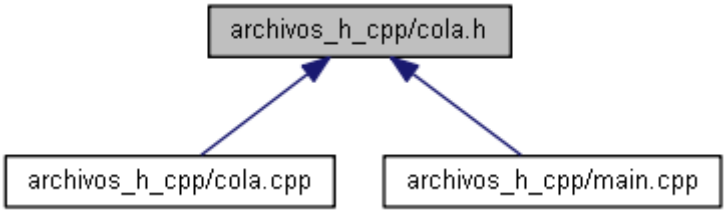


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Clases

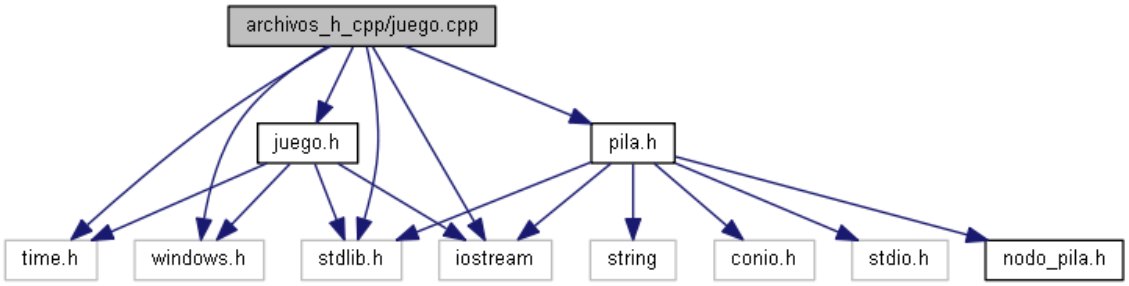
- class cola

Estructura lineal para almacenar los números de la cinta.

Referencia del Archivo archivos_h_cpp/juego.cpp

```
#include <stdlib.h>
#include <time.h>
#include <iostream>
#include <windows.h>
#include "pila.h"
#include "juego.h"
```

Dependencia gráfica adjunta para juego.cpp:



Referencia del Archivo archivos_h_cpp/juego.h

```
#include <stdlib.h>
#include <time.h>
#include <iostream>
#include <windows.h>
```

Dependencia gráfica adjunta para juego.h:

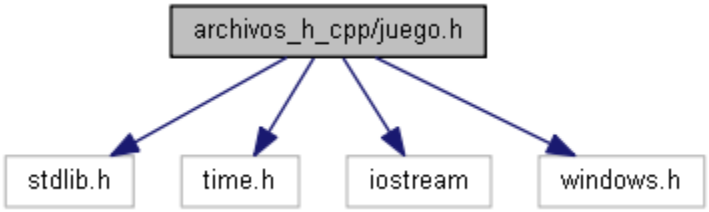
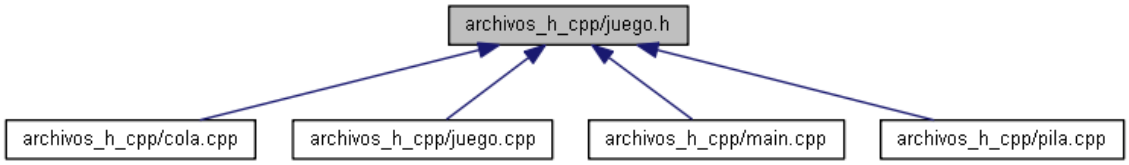


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Clases

- class juego

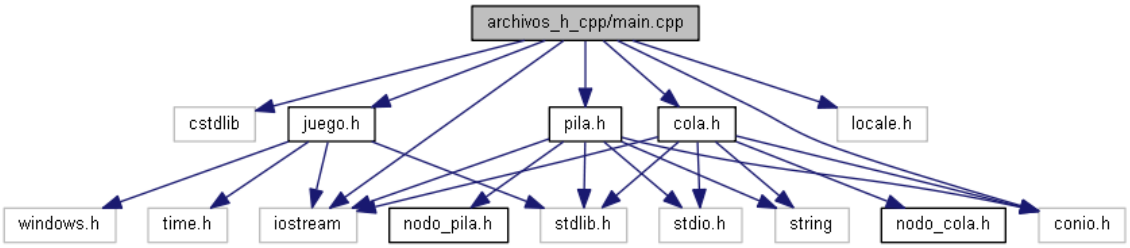
Clase que proporciona métodos estáticos, para el funcionamiento del juego.

Referencia del Archivo archivos_h_cpp/main.cpp

cpp. Contiene la funcion que ejecuta el programa.

```
#include <cstdlib>
#include <iostream>
#include "juego.h"
#include "pila.h"
#include "cola.h"
#include <locale.h>
#include <conio.h>
```

Dependencia gráfica adjunta para main.cpp:



Funciones

- int main (int argc, char *argv[])
  Funcion que representa el punto de inicio de ejecución del programa.

Descripción detallada

cpp. Contiene la funcion que ejecuta el programa.

Autor:

Jorge Abad

Definición en el archivo main.cpp.

Documentación de las funciones

int main (int *argc*, char * *argv*[])

Funcion que representa el punto de inicio de ejecución del programa.

- Instancia un objeto de la clase cola, para reperesentar la cinta.
- Instancia 4 objetos de la clase pila, para representar los 4 montones.
- Llama a los metodos de la clase juego, para realizar y obtener las operaciones.
- Muestra la interfaz grafica que le aparece al usuario mostrando un menú con opciines.
- Crea las variables locales necesarias para interactuar con el usuario, como mensajes de aviso y opciones elegidas.

Variables:

- valorReferencia: numero entero que representa el numero máximo de elementos que puede haber en un montón
- montonSeleccionado: numero entero que representa el monton seleccionado por el usuario
- monton1, monton2, monton3, monton4: objetos de la clase pila representan los 4 montones
- cinta: objeto de la clase cola, representará la cinta de numeros
- op: variable char almacena el símbolo de la operción generada
- avisoSistema: variable string Almacena cualquier aviso de error o advertencia, para el usuario al interactuar con el programa
- finJuego: variable string Almacena una cadena con un mensaje de fin de programa, ganar o perder
- opcion: variable char que almacena la opción seleccionada por el usuario. si es 's' salimos del programa
- numeroCinta variable shor almacena el primer numero de la cinta, el que hay que introducir en los montones.
- cima: variable short Almacena el número que es cima del monton seleccionado con el que operaremos.
- resultado: variable short que almacena el resultado de la operacion entre el primero de la cinta y el último del montón
- montonStd []: array char almacena el monton elegido para mostrarlo al usuario por la pantalla valores ' ', '1', '2', '3', '4'

Definición en la línea 25 del archivo main.cpp.

```
26 {
27
28     setlocale(LC_ALL, "spanish");  /*poder representar acentos*/
29     juego::iniciarSemilla(); /*iniciamos la semilla para generar los números
aleatorios*/
32     const short valorReferencia=10; //No se permitirá mas de 10 elementos en un
montón
33
34
36     short montonSeleccionado=0;//inicialmente no hay ningún montón seleccionado, 0.
38     /*Creamos los montones en donde almacenaremos los numeros, instanciamos objetos
de la clase pila*/
39     pila monton1;
40     pila monton2;
41     pila monton3;;
42     pila monton4;
43
45     /*Instanciamos un objeto de la clase cola, será la cinta dónde iran apareciendo
los nuevos números*/
46     cola cinta;
47
48     /*Mediante el método llenarPila(), de la clase pila, llenamos los 4 montones, con
numeros aleatorios y con una longitud aleatoria
49     para cada pila, los numeros y la longitusd estarán entre 1 y 9*/
50     monton1.llenarPila();
51     monton2.llenarPila();
52     monton3.llenarPila();
53     monton4.llenarPila();
54
55     /*Llenamo la cinta con 5 numeros mediante el método llenarCola() de la clase
cola*/
56     cinta.llenarCola();
57
59     char op=' '; //Almacena el símbolo de la operación
61     string avisoSistema=""; //Almacena cualquier aviso de error o advertencia, para
el usuario al interactuar con el programa.
63     string finJuego="fin del programa"; //Avisa del fin del programa.
65     char opcion='0';//variable que almacena la opción seleccionada por el
usuario.Cero no hay opción.
67     short numeroCinta=0; // almacena el primer numero de la cinta, el que hay que
introducir en los montones.Cero no hay número sacado de la cinta.
```

```
69     short cima;//Almacena el número que es cima del monton seleccionado.
71     short resultado=0;//Almacena el resultado de la operación inicialmente 0.
72
73 /*Mientras la opción seleccionada no sea 's' se limpia la pantalla y de nuevo se
muestra la situación actual
74    y valor actual de las variables oportunas. Mostrará situación actual de la cinta,
de los montones, el monton
75    elegido, la operación generada, un menú de opciones, y un mensaje de aviso para
interactuar con el juego.*/
76 while (opcion != 's')
77     {
78
79    system ("cls"); //limpiamos la pantalla.
80
81 /*Cabecera presentación.*/
82 cout<<endl<<"    #------------------|* NUMBERIS  -  PRACTICA1 *|------------------
#"<<endl;
83
84 system("color 0B");//color del fondo y del texto de la consola.
85
86 numeroCinta=cinta.primero en cola(); //Almacenamos el el primer número de la cinta.
87
88 /*Mostramos por pantalla todos los elementos de la cinta, método mostrar() de la
clase cola*/
89 cout<<" Cinta numérica: ";
90 cout<<"\n --------------";
91 /*Mostramos por pantalla el primer y el resto de números de la cinta, cola*/
92     cout<<"       Primero: "<<numeroCinta
93     <<"     Resto: ";cinta.mostrarResto(); //Mostramos el primer numero de la cinta.
94     cout<<"\n";
95
96
97     /*Mostramos por pantalla el contenido de los cuatros montones mediante el metodo
imprimir() de la
98        clase pila, para cada objeto pila que ha sido instanciado.*/
99     cout<<"\n MONTóN 1-> ";monton1.imprimir();
100     cout<<"\n";
101
102     cout<<" MONTóN 2-> ";monton2.imprimir();
103     cout<<"\n";
104
105     cout<<" MONTóN 3-> ";monton3.imprimir();
106     cout<<"\n";
107
108     cout<<" MONTóN 4-> ";monton4.imprimir();
109     cout<<"\n";
111  char montonStd [] =" ";//Muestra por pantalla el monton elegido. Inicialmente no
hay monton seleccionado.
112 montonStd [0]=(char)(montonSeleccionado+48);//convertimos a char el valor de
montonSeleccionado y lo almacenamos en la primera posicion del array
113 if (montonSeleccionado==0){//Si montonSeleccionado es igual a cero, no hay monton
seleccionado
114     montonStd[0]=' ';//la primera posición es vacía.
115 }
116
117 /*Mostramos información, con el menú, los valores seleccionados y generados para
cada vez que el usuario
118 seleccina una opción y pulsa intro. Se mostrará el montón elegido por el usuario y
la operación que se genera*/
119 cout<<"\n-------------------------------------------------------------------------\n";
120 cout<<"         OPERACION-> "<<op<<"     MONTON SELECCIONADO-> "<<montonStd<<"\n";
121 cout<<"-------------------------------------------------------------------------\n";
122
123 /*Saca por pantalla las opciones del menú.  */
124 cout<<"     0- Generar operación "<<endl<<endl;
125 cout<<"     1- Colocar el nº en el montón 1"<<endl<<
126 "     2- Colocar el nº en el montón 2"<<endl<<
127 "     3- Colocar el nº en el montón 3"<<endl<<
128 "     4- Colocar el nº en el montón 4"<<endl<<
129 "\n     s- Salir "<<endl;
130
```

```
131 /*Mostramos un aviso del sistema con otro color de fuente, verde*/
132 HANDLE console = GetStdHandle(STD_OUTPUT_HANDLE);
133 cout<<"Mensaje del Sistema: ";
134 SetConsoleTextAttribute(console, 10);/*Fuente de color verde para el siguiente
cout<<*/
135 cout<<avisoSistema;//Mostrara el contenido de la variable para advertir de cualquier
error.
136 SetConsoleTextAttribute(console, 11);/*volvemos a poner letras en azul claro*/
137
138 cout<<"\n Eliga una opción: ";
139 cin>>opcion; //leer la opción elegida y lo almacenamos en la variable opción
140
141 system("cls");//despues de leer la opción elegida limpiamos pantalla
142 system("color 0A");  //color del fondo y del texto de la consola.
143
144
145
146 /*En función del valor de opcion y de valores anteriores de las variables op y
montonSeleccionado,
147 seleccionaremos un monton o generaremos una operación o realizamos la operacion
148 con primero de la cinta y la cima de un monton**/
149     switch (opcion)
150     {
151         /*Si opcion es '0', si se puede, generaremos una operación**/
152         case '0':{
153             if(op!=' '){
154                 avisoSistema="Ya hay operación generada no puede solicitar otra, \n
pulse una tecla diferente a las opciones";
155                 break;
156             }
157             /*Solo generamos la operación si hay un monton seleccionado, si es igual
a cero avisamos*/
158             if (montonSeleccionado==0){
159             avisoSistema="No se puede generar una operación, primero ha de elegir un
montón";
160             /*Si el hay un monton seleccionado, montonSeleccionado es distinto de
cero ,obtenemos una operación*/
161             }else{
162             op=juego::obtenerOperacion();//Obtenemos un operación aleatoria y la
almacenamos.
163             avisoSistema="Pulse una tecla diferente a las opciones para continuar.";
164             }
165             break;
166             }
167
168         /*Opciones para seleccionar un monton*/
169
170           /*opcion es igual a '1'*/
171            case '1':{
172            /*Comprobamos que op sea igual a vacio, no halla operación generada
previamente y que el monton 1 no esté vacio*/
173                 if (monton1.es pila vacia() && op==' '){
174             avisoSistema="Montón 1 vacio no lo puede usar, seleccione
otro.";//Avisamos que ese montón no puede usarse.
175             break;
176            }
177
178             /*Si no hay operacion generada, op tiene valor vacio, podemos elegir
montón*/
179             if(op==' '){
180             montonSeleccionado=1;/*almacenamos el valor 1 en montonSeleccionado*/
181             avisoSistema="Genere una operacion o cambie de montón";/*podemos cambiar
de montón al no haber operación*/
182
183             }else{/*Hay una operacion generada, var op diferente vacío, no podemos
intentar cambiar de montón*/
184                 avisoSistema="Ya hay operación generada no puede cambiar de montón,
\n pulse una tecla diferente a las opciones para continuar";
185             }
186
187                 break;
```

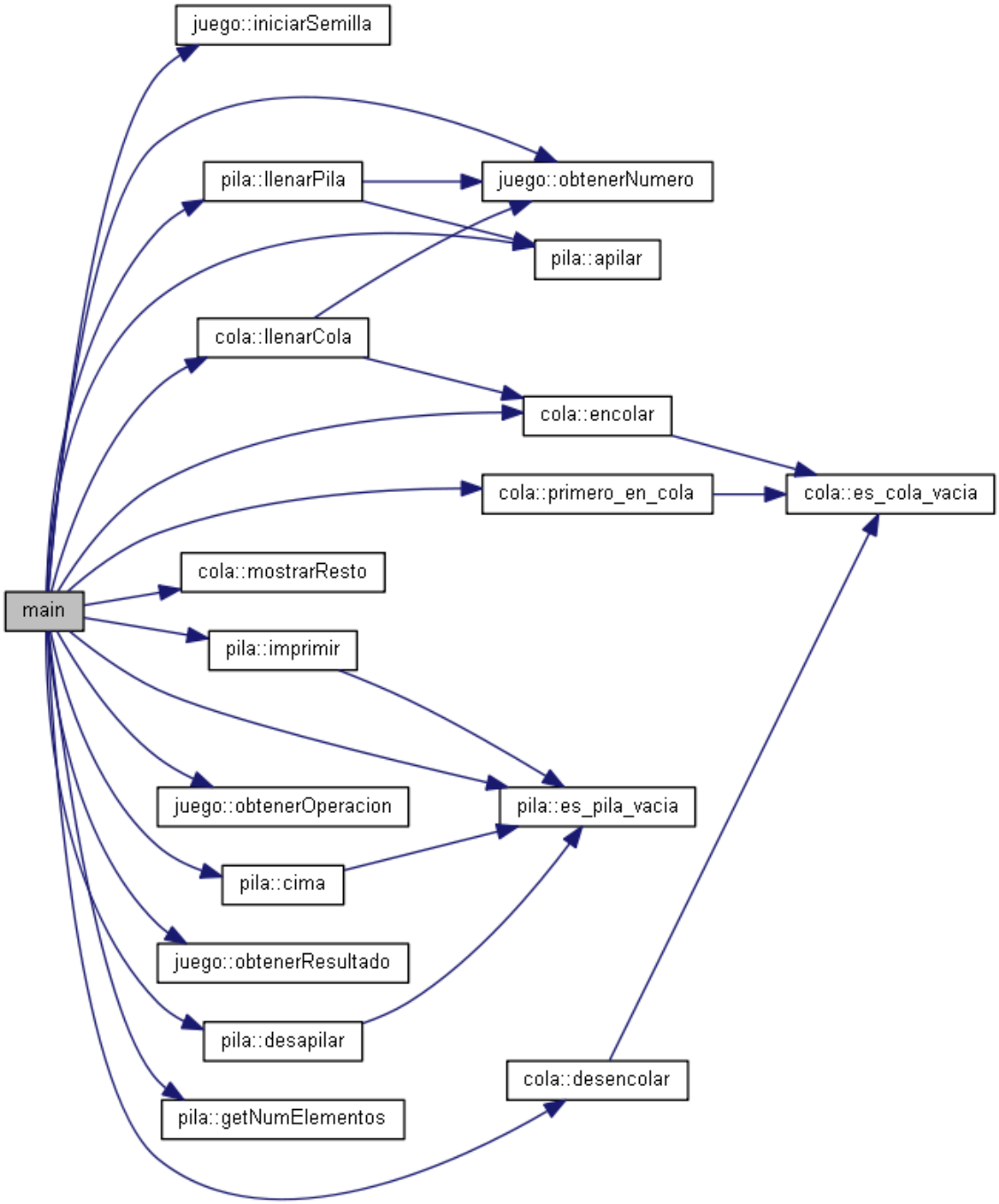
```
188
189                 }
190             /*opcion es igual a '2'*/
191             case '2':{
192             /*Comprobamos que op sea igual a vacio, no halla operación generada
previamente y que el monton 2 no esté vacio*/
193             if (monton2.es pila vacia() && op==' '){
194              avisoSistema="Monton 2 vacio no lo puede usar, seleccione
otro.";//Avisamos que ese montón no puede usarse.
195              break;
196             }
197              /*Si no hay operacion generada, op tiene valor vacio, podemos elegir
montón*/
198              if(op==' '){
199             montonSeleccionado=2;/*almacenamos el valor 2 en montonSeleccionado*/
200             avisoSistema="Genere una operacion o cambie de montón";/*podemos cambiar
de montón al no haber operación*/
201
202              }else{ /*Hay una operacion generada, var op diferente vacío, no podemos
intentar cambiar de montón*/
203                 avisoSistema="Ya hay operación generada no puede cambiar de montón,
\n pulse una tecla diferente a las opciones para continuar";
204              }
205                 break;
206                 }
207
208             /*opcion es igual a '3'*/
209             case '3':{
210
211               if (monton3.es pila vacia() && op==' '){
212             avisoSistema="Monton 3 vacio no lo puede usar, seleccione
otro.";//Avisamos que ese montón no puede usarse.
213              break;
214             }
215                 /*Si no hay operacion generada, op tiene valor vacio, podemos elegir
montón*/
216              if(op==' '){
217                 montonSeleccionado=3;/*almacenamos el valor 3 en
montonSeleccionado*/
218                 avisoSistema="Genere una operacion o cambie de montón";
219                 }else{
220                 avisoSistema="Ya hay operación generada no puede cambiar de montón,
\n pulse una tecla diferente a las opciones para continuar";
221              }
222                 break;
223                 }
224
225             /*opcion es igual a '4'*/
226             case '4':{
227
228               if (monton4.es pila vacia() && op==' '){
229              avisoSistema="Monton 4 vacio no lo puede usar, seleccione
otro.";//Avisamos que ese montón no puede usarse.
230              break;
231             }
232               /*Si no hay operacion generada, op tiene valor vacio, podemos elegir
montón*/
233              if(op==' '){
234                 montonSeleccionado=4;
235                 avisoSistema="Genere una operacion o cambie de montón";
236
237                  }else{
238                  avisoSistema="Ya hay operación generada no puede cambiar de montón,
\n pulse una tecla diferente a las opciones para continuar";
239              }
240                 break;
241              }
242                 /*opcion es igual a otro valor diferente de '0','1','2','3','4'*/
243                 default:{
244                     /*si no hay monton ni operación, montonSeleccionado, op, son
cero o vacio respectivamente, salimos*/
```

```
245                     if (montonSeleccionado==0 || op==' '){
246                         avisoSistema="No hay montón seleccionado o una operación
creada eliga otra opción";
247
248
249                     }else{
250                     /*Hay un montón seleccionado, montonSeleccionado es diferente de
cero y una operación, op es diferente de ' '*/
251                     /*Para cada montón seleccionado, determinado por el valor
montonSeleccionado, actuamos*/
252                     switch(montonSeleccionado){
253
254                         case 1:{
255                             /*la variable cima toma el valor de la cima del montón
1*/
256                             cima=monton1.cima();
257                             /*Pasamos los valores,un símbolo de operación, y dos
números, para operar, y obtener un resultado */
258                             resultado=juego::obtenerResultado(op,numeroCinta,cima);
259                             /*Si el resultado es menor que un valor de referencia
(10)*/
260                             if (resultado<valorReferencia){
261                                 /*Colocamos el número de la cinta en la cima del
montón apilamos */
262                                 monton1.apilar(numeroCinta);
263                                 avisoSistema=" Elemento colocado en montón 1";
264                             }else{
265                                 /*En caso contrario procedemos a quitar la cima del
montón, desapilar*/
266                                 monton1.desapilar();
267                                 avisoSistema=" La cima del montón 1 ha sido
eliminada ";
268                             }
269                             /*Una vez desapilado o apilado comprobamos si el montón
ha llegado al limite*/
270                                 if (monton1.getNumElementos()==10){
271                                 /*Al llegar al limite alamacenamos un mensje en la
var finJuego*/
272                                 finJuego="PERDISTE MONTON 1 LLENO";
273                                 /*Provocamos la salida del while, asignando el valor
's' a la var opcion */
274                                 opcion='s';
275                             }
276
277                             break;
278                         }
279                         case 2:{
280
281                                 cima=monton2.cima();
282                             resultado=juego::obtenerResultado(op,numeroCinta,cima);
283                             if (resultado<valorReferencia){
284                                 monton2.apilar(numeroCinta);
285                                 avisoSistema=" Elemento colocado en montón 2";
286                             }else{
287                                 monton2.desapilar();
288                                 avisoSistema=" La cima del montón 2 ha sido
eliminada ";
289                             }
290                                 if (monton2.getNumElementos()==10){
291                                 finJuego="PERDISTE MONTON 2 LLENO";
292                                 opcion='s';
293                             }
294
295
296                             break;
297                         }
298                         case 3:{
299
300                             cima=monton3.cima();
301                             resultado=juego::obtenerResultado(op,numeroCinta,cima);
302                             if (resultado<valorReferencia){
```

```
303                                 monton3.apilar(numeroCinta);
304                                 avisoSistema=" Elemento colocado en montón 3";
305                             }else{
306                                 monton3.desapilar();
307                                 avisoSistema=" La cima del montón 3 ha sido
eliminada ";
308                             }
309                                 if (monton3.getNumElementos()==10){
310                                 finJuego="PERDISTE MONTON 3 LLENO";
311                                 opcion='s';
312                             }
313
314                             break;
315                         }
316                         case 4:{
317
318                             cima=monton4.cima();
319                             resultado=juego::obtenerResultado(op,numeroCinta,cima);
320                             if (resultado<valorReferencia){
321                                 monton4.apilar(numeroCinta);
322                                     avisoSistema=" Elemento colocado en montón 4";
323                             }else{
324                                 monton4.desapilar();
325                                 avisoSistema=" La cima del montón 4 ha sido
eliminada ";
326                             }
327                                 if (monton4.getNumElementos()==10){
328                                 finJuego="PERDISTE MONTON 4 LLENO";
329                                 opcion='s';
330                             }
331
332                             break;
333                         }
334
335                     }
336
337             /*Despues de actuar sobre un montón, comprobamos si todos los montones
están ya vacios*/
338
if(monton1.es pila vacia()&&monton2.es pila vacia()&&monton3.es pila vacia()&&monton4.es
 pila vacia())  {
339                         /*Al estar todos vacíos el juego se gana almacenamos el
valor adecuado en la var finJuego*/
340                         finJuego="Ganaste el juego todos los montones vacios";
341                         /*Provocaremos la salida del while asignando el valor 's' a
la opción*/
342                         opcion='s';
343                     }else{
344                         /*Si no están vacios reiniciamos las variables con los
valores adecuados*/
345                         avisoSistema=avisoSistema+".\n Seleccione un montón para
continuar jugando";
346                         montonSeleccionado=0;
347                         op=' ';
348                         /*Actuamos sobre la cinta, quitando y poniendo número*/
349                         /*Quitamos el primero de la cinta, desencolamos*/
350                         cinta.desencolar();
351                         /*Añadimos un nuevo numero a la cinta por el final,
encolamos.*/
352                         cinta.encolar(juego::obtenerNumero());
353                     }
354
355
356
357
358         }
359                 }
360
361
362 }
363
```

```
364
365
366
367
368
369
370     }
371     system("cls");
372
373
374
375     //Final del programa
376     cout<<"##################################################"<<endl<<
377             finJuego<<"*"<<endl<<
378           "##################################################"<<endl;
379     _getche();
380
381
382 }
```

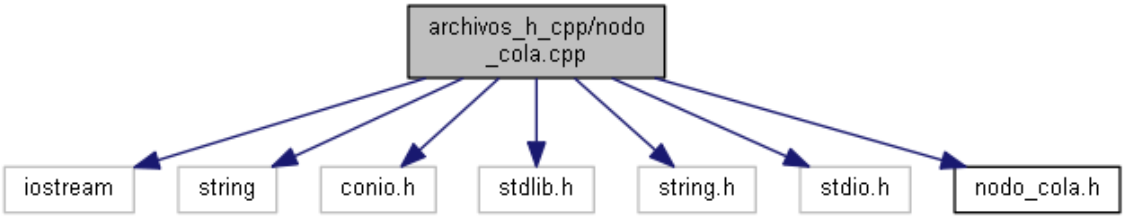
Gráfico de llamadas para esta función:


juego::iniciarSemilla
pila::llenarPila
juego::obtenerNumero
pila::apilar
cola::llenarCola
cola::encolar
cola::primero_en_cola
cola::es_cola_vacia
cola::mostrarResto
main
pila::imprimir
juego::obtenerOperacion
pila::es_pila_vacia
pila::cima
juego::obtenerResultado
pila::desapilar
cola::desencolar
pila::getNumElementos

Referencia del Archivo archivos_h_cpp/nodo_cola.cpp

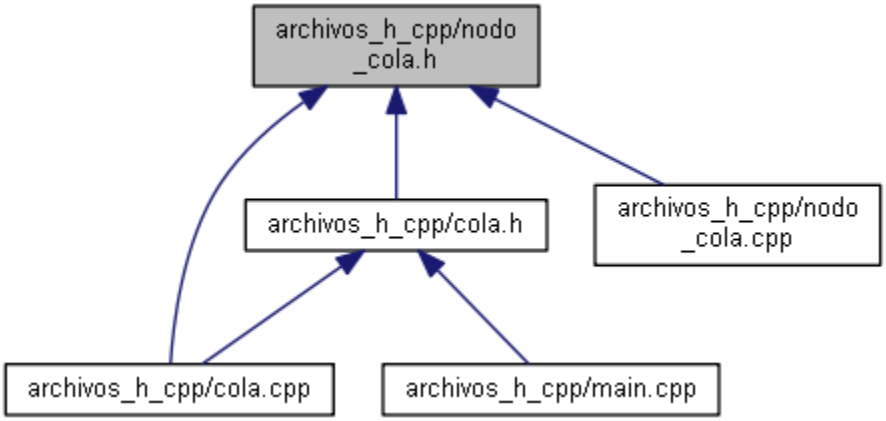
```
#include <iostream>
#include <string>
#include <conio.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "nodo_cola.h"
```

Dependencia gráfica adjunta para nodo_cola.cpp:



Referencia del Archivo archivos_h_cpp/nodo_cola.h

Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Clases

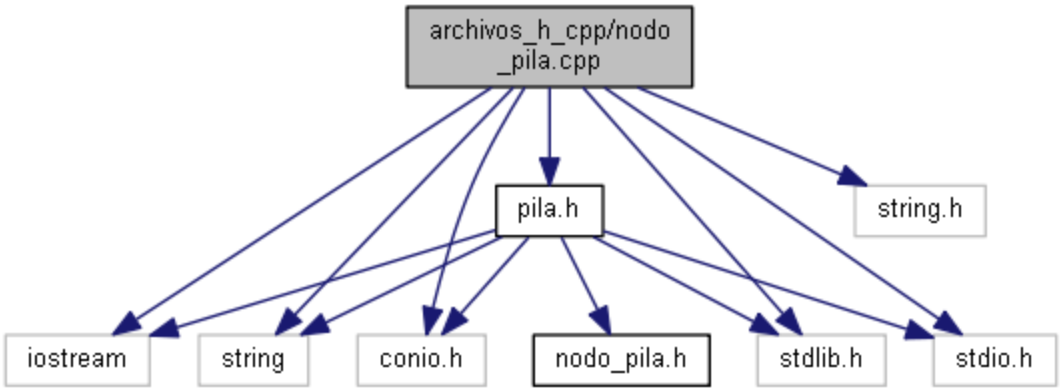
- class nodo_cola

Representa una celda de la cola. Almacena cada uno de los números que componenen la cinta.

Referencia del Archivo archivos_h_cpp/nodo_pila.cpp

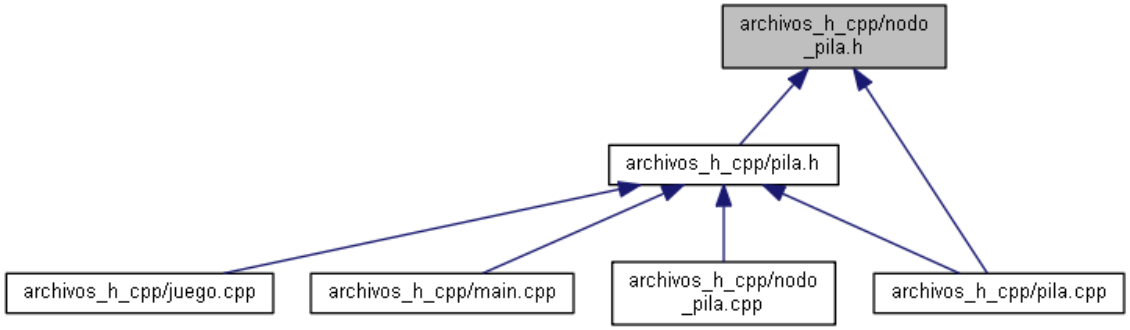
```
#include <iostream>
#include <string>
#include <conio.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "pila.h"
```

Dependencia gráfica adjunta para nodo_pila.cpp:



Referencia del Archivo archivos_h_cpp/nodo_pila.h

Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Clases

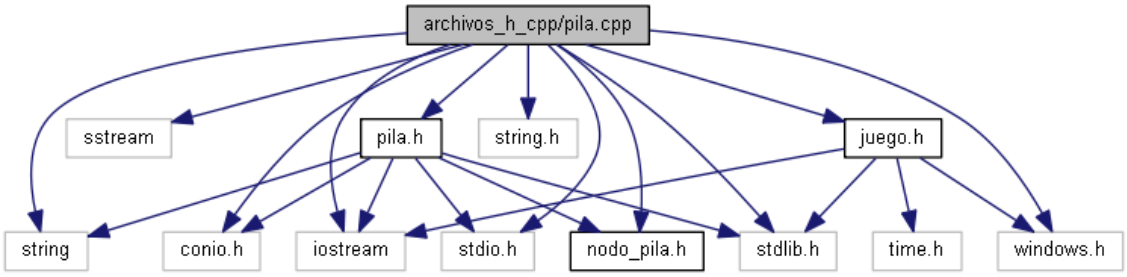
- class nodo_pila

Representa una celda de la pila. Almacena cada uno de los números que componene un montón

Referencia del Archivo archivos_h_cpp/pila.cpp

```
#include <iostream>
#include <sstream>
#include <string>
#include <conio.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <windows.h>
#include "pila.h"
#include "nodo_pila.h"
#include "juego.h"
```

Dependencia gráfica adjunta para pila.cpp:



Referencia del Archivo archivos_h_cpp/pila.h

```
#include <iostream>
#include <string>
#include <conio.h>
#include <stdlib.h>
#include <stdio.h>
#include "nodo_pila.h"
```

Dependencia gráfica adjunta para pila.h:

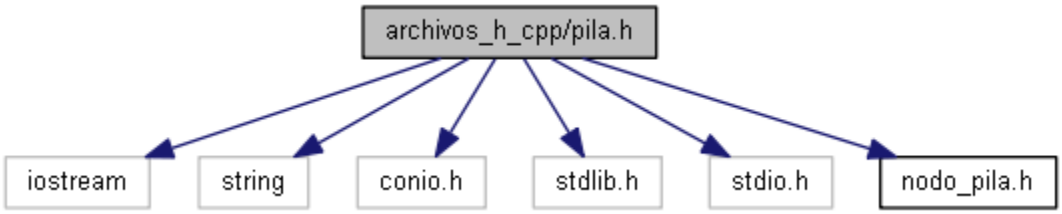
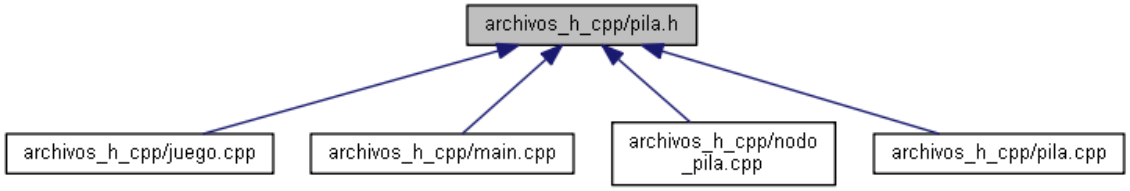


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Clases

- class pila

Estructura lineal para almacenar los números de los montones.

Índice