COLA:

|  |
| --- |
| #include <iostream> |
|  | #include <windows.h> |
|  | using namespace std; |
|  |  |
|  |  |
|  | /\* Estructura de los nodos de la cola |
|  | ------------------------------------------------------------------------\*/ |
|  | struct nodo |
|  | { |
|  | int nro; |
|  | struct nodo \*sgte; |
|  | }; |
|  |  |
|  |  |
|  | /\* Estructura de la cola |
|  | ------------------------------------------------------------------------\*/ |
|  | struct cola |
|  | { |
|  | nodo \*delante; |
|  | nodo \*atras ; |
|  | }; |
|  |  |
|  |  |
|  | /\* Encolar elemento |
|  | ------------------------------------------------------------------------\*/ |
|  | void encolar( struct cola &q, int valor ) |
|  | { |
|  | struct nodo \*aux = new(struct nodo); |
|  |  |
|  | aux->nro = valor; |
|  | aux->sgte = NULL; |
|  |  |
|  | if( q.delante == NULL) |
|  | q.delante = aux; // encola el primero elemento |
|  | else |
|  | (q.atras)->sgte = aux; |
|  |  |
|  | q.atras = aux; // puntero que siempre apunta al ultimo elemento |
|  |  |
|  | } |
|  |  |
|  | /\* Desencolar elemento |
|  | ------------------------------------------------------------------------\*/ |
|  | int desencolar( struct cola &q ) |
|  | { |
|  | int num ; |
|  | struct nodo \*aux ; |
|  |  |
|  | aux = q.delante; // aux apunta al inicio de la cola |
|  | num = aux->nro; |
|  | q.delante = (q.delante)->sgte; |
|  | delete(aux); // libera memoria a donde apuntaba aux |
|  |  |
|  | return num; |
|  | } |
|  |  |
|  | /\* Mostrar Cola |
|  | ------------------------------------------------------------------------\*/ |
|  | void muestraCola( struct cola q ) |
|  | { |
|  | struct nodo \*aux; |
|  |  |
|  | aux = q.delante; |
|  |  |
|  | while( aux != NULL ) |
|  | { |
|  | cout<<" "<< aux->nro ; |
|  | aux = aux->sgte; |
|  | } |
|  | } |
|  |  |
|  | /\* Eliminar todos los elementos de la Cola |
|  | ------------------------------------------------------------------------\*/ |
|  | void vaciaCola( struct cola &q) |
|  | { |
|  | struct nodo \*aux; |
|  |  |
|  | while( q.delante != NULL) |
|  | { |
|  | aux = q.delante; |
|  | q.delante = aux->sgte; |
|  | delete(aux); |
|  | } |
|  | q.delante = NULL; |
|  | q.atras = NULL; |
|  |  |
|  | } |
|  |  |
|  | /\* Menu de opciones |
|  | ------------------------------------------------------------------------\*/ |
|  | void menu() |
|  | { |
|  | cout<<"\n\t CREANDO COLA, ¿QUÉ DESEA HACER?\n\n"; |
|  | cout<<" 1. ENCOLAR "<<endl; |
|  | cout<<" 2. DESENCOLAR "<<endl; |
|  | cout<<" 3. MOSTRAR COLA "<<endl; |
|  | cout<<" 4. VACIAR COLA "<<endl; |
|  | cout<<" 5. SALIR "<<endl; |
|  |  |
|  | cout<<"\n INGRESE OPCION: "; |
|  | } |
|  |  |
|  | /\* Funcion Principal |
|  | ------------------------------------------------------------------------\*/ |
|  | int main() |
|  | { |
|  | struct cola q; |
|  |  |
|  | q.delante = NULL; |
|  | q.atras = NULL; |
|  |  |
|  |  |
|  | int dato; // numero a encolar |
|  | int op; // opcion del menu |
|  | int x ; // numero que devuelve la funcon pop |
|  |  |
|  | system("color 0b"); |
|  |  |
|  | do |
|  | { |
|  | menu(); cin>> op; |
|  |  |
|  | switch(op) |
|  | { |
|  | case 1: |
|  |  |
|  | cout<< "\n NUMERO A ENCOLAR: "; cin>> dato; |
|  | encolar( q, dato ); |
|  | cout<<"\n\n\t\tNumero " << dato << " encolado...\n\n"; |
|  | break; |
|  |  |
|  |  |
|  | case 2: |
|  |  |
|  | x = desencolar( q ); |
|  | cout<<"\n\n\t\tNumero "<< x <<" desencolado...\n\n"; |
|  | break; |
|  |  |
|  |  |
|  | case 3: |
|  |  |
|  | cout << "\n\n MOSTRANDO COLA\n\n"; |
|  | if(q.delante!=NULL) muestraCola( q ); |
|  | else cout<<"\n\n\tCola vacia...!"<<endl; |
|  | break; |
|  |  |
|  |  |
|  | case 4: |
|  |  |
|  | vaciaCola( q ); |
|  | cout<<"\n\n\t\tHecho...\n\n"; |
|  | break; |
|  |  |
|  |  |
|  | default: |
|  |  |
|  | cout<<"\n\n\tEres un estúpido!!!\n\n"<<endl; |
|  | break; |
|  |  |
|  | } |
|  |  |
|  | cout<<endl<<endl; |
|  | system("pause"); system("cls"); |
|  |  |
|  | }while(op!=5); |
|  |  |
|  |  |
|  | return 0; |
|  | } |

COLA DE PRIORIDAD:

|  |
| --- |
| #include <iostream> |
|  | #include <windows.h> |
|  | using namespace std; |
|  |  |
|  | // Prioridad de la Cola // |
|  |  |
|  | struct prioridad |
|  | { |
|  | int prio; |
|  | struct cola \*sacy; |
|  | }; |
|  |  |
|  |  |
|  | /\* Estructura de los nodos de la cola |
|  | ------------------------------------------------------------------------\*/ |
|  | struct nodo |
|  | { |
|  | int nro; |
|  | struct nodo \*sgte; |
|  | }; |
|  |  |
|  |  |
|  | /\* Estructura de la cola |
|  | ------------------------------------------------------------------------\*/ |
|  | struct cola |
|  | { |
|  | nodo \*delante; |
|  | nodo \*atras ; |
|  | }; |
|  |  |
|  |  |
|  | /\* Encolar elemento |
|  | ------------------------------------------------------------------------\*/ |
|  | void encolar( struct cola &q, int valor ) |
|  | { |
|  | struct nodo \*aux = new(struct nodo); |
|  |  |
|  | aux->nro = valor; |
|  | aux->sgte = NULL; |
|  |  |
|  | if( q.delante == NULL) |
|  | q.delante = aux; // encola el primero elemento |
|  | else |
|  | (q.atras)->sgte = aux; |
|  |  |
|  | q.atras = aux; // puntero que siempre apunta al ultimo elemento |
|  |  |
|  | } |
|  |  |
|  | /\* Desencolar elemento |
|  | ------------------------------------------------------------------------\*/ |
|  | int desencolar( struct cola &q ) |
|  | { |
|  | int num ; |
|  | struct nodo \*aux ; |
|  |  |
|  | aux = q.delante; // aux apunta al inicio de la cola |
|  | num = aux->nro; |
|  | q.delante = (q.delante)->sgte; |
|  | delete(aux); // libera memoria a donde apuntaba aux |
|  |  |
|  | return num; |
|  | } |
|  |  |
|  | /\* Mostrar Cola |
|  | ------------------------------------------------------------------------\*/ |
|  | void muestraCola( struct cola q ) |
|  | { |
|  | struct nodo \*aux; |
|  |  |
|  | aux = q.delante; |
|  |  |
|  | while( aux != NULL ) |
|  | { |
|  | cout<<" "<< aux->nro ; |
|  | aux = aux->sgte; |
|  | } |
|  | } |
|  |  |
|  | /\* Eliminar todos los elementos de la Cola |
|  | ------------------------------------------------------------------------\*/ |
|  | void vaciaCola( struct cola &q) |
|  | { |
|  | struct nodo \*aux; |
|  |  |
|  | while( q.delante != NULL) |
|  | { |
|  | aux = q.delante; |
|  | q.delante = aux->sgte; |
|  | delete(aux); |
|  | } |
|  | q.delante = NULL; |
|  | q.atras = NULL; |
|  |  |
|  | } |
|  |  |
|  | /\* Menu de opciones |
|  | ------------------------------------------------------------------------\*/ |
|  | void menu() |
|  | { |
|  | cout<<"\n\t CREANDO COLA DE PRIORIDAD, ¿QUÉ DESEA HACER?\n\n"; |
|  | cout<<" 1. ENCOLAR "<<endl; |
|  | cout<<" 2. DESENCOLAR "<<endl; |
|  | cout<<" 3. MOSTRAR COLA "<<endl; |
|  | cout<<" 4. VACIAR COLA "<<endl; |
|  | cout<<" 5. SALIR "<<endl; |
|  |  |
|  | cout<<"\n INGRESE OPCION: "; |
|  | } |
|  |  |
|  | /\* Funcion Principal |
|  | ------------------------------------------------------------------------\*/ |
|  | int main() |
|  | { |
|  | int f=1; |
|  | prioridad G; |
|  | G.prio=f; |
|  | struct cola \*q; |
|  | G.sacy=q; |
|  |  |
|  |  |
|  | q->delante = NULL; |
|  | q->atras = NULL; |
|  |  |
|  |  |
|  | int dato; // numero a encolar |
|  | int op; // opcion del menu |
|  | int x ; // numero que devuelve la funcon pop |
|  |  |
|  | system("color 0b"); |
|  |  |
|  | do |
|  | { |
|  | menu(); cin>> op; |
|  |  |
|  | switch(op) |
|  | { |
|  | case 1: |
|  |  |
|  | cout<< "\n NUMERO A ENCOLAR: "; cin>> dato; |
|  | encolar( \*q, dato ); |
|  | cout<<"\n\n\t\tNumero " << dato << " encolado...\n\n"; |
|  | break; |
|  |  |
|  |  |
|  | case 2: |
|  |  |
|  | x = desencolar( \*q ); |
|  | cout<<"\n\n\t\tNumero "<< x <<" desencolado...\n\n"; |
|  | break; |
|  |  |
|  |  |
|  | case 3: |
|  |  |
|  | cout << "\n\n MOSTRANDO COLA\n\n"; |
|  | if(q->delante!=NULL) muestraCola( \*q ); |
|  | else cout<<"\n\n\tCola vacia...!"<<endl; |
|  | break; |
|  |  |
|  |  |
|  | case 4: |
|  |  |
|  | vaciaCola( \*q ); |
|  | cout<<"\n\n\t\tHecho...\n\n"; |
|  | break; |
|  |  |
|  | } |
|  |  |
|  | cout<<endl<<endl; |
|  | system("pause"); system("cls"); |
|  |  |
|  | }while(op!=5); |
|  |  |
|  |  |
|  | return 0; |
|  | } |

LISTA:

|  |
| --- |
| #include <iostream> |
|  | #include <stdlib.h> |
|  |  |
|  | using namespace std; |
|  |  |
|  | struct node |
|  | { |
|  | int val; |
|  | struct node \*next; |
|  | }; |
|  |  |
|  | void crear(struct node \*head) |
|  | { |
|  | struct node \*prev, \*cur; |
|  | cout << "Cuantos nodos desea:" << endl; |
|  | int n; |
|  | cin >> n; |
|  | head->val=1; |
|  | prev=head; |
|  | if (n>1){ |
|  | for (int i=2; i<=n; i++){ |
|  | cur=(struct node\*)malloc(sizeof(struct node)); |
|  | cur->val=i; |
|  | prev->next=cur; |
|  | prev=cur; |
|  | } |
|  | } |
|  | prev->next=NULL; |
|  | }; |
|  |  |
|  | void impr(struct node \*head) |
|  | { |
|  | cout << "IMPRIMIENDO:" << endl; |
|  | struct node \*t; |
|  | t=head; |
|  | while (t!=NULL){ |
|  | cout << t->val << endl; |
|  | t=t->next; |
|  | } |
|  | }; |
|  |  |
|  | struct node \*elim(struct node \*head) |
|  | { |
|  | cout << "Quitando nodo:" << endl << "Retire un termino existente:" << endl; |
|  | int key; |
|  | cin >> key; |
|  | if (head->val==key){ |
|  | struct node \*temp; |
|  | temp=head->next; |
|  | free(head); |
|  | head=temp; |
|  | return head; |
|  | } |
|  | else { |
|  | struct node \*prev, \*cur; |
|  | prev=head; |
|  | cur=head->next; |
|  | while(cur!=NULL){ |
|  | if (cur->val==key){ |
|  | prev->next=cur->next; |
|  | free(cur); |
|  | break; |
|  | } |
|  | prev=cur; |
|  | cur=cur->next; |
|  | } |
|  | return head; |
|  | } |
|  | }; |
|  |  |
|  | int main() |
|  | { |
|  | struct node \*head; |
|  | head=(struct node\*)malloc(sizeof(struct node)); |
|  | crear(head); |
|  | impr(head); |
|  | head=elim(head); |
|  | impr(head); |
|  | return 0; |
|  | } |

LISTA DOBLE:

|  |
| --- |
| #include <iostream> |
|  | #include <stdlib.h> |
|  |  |
|  | using namespace std; |
|  |  |
|  | struct node |
|  | { |
|  | int val; |
|  | struct node \*next, \*ante; |
|  | }; |
|  |  |
|  | void crear(struct node \*head) |
|  | { |
|  | struct node \*prev, \*cur, \*vir; |
|  | cout << "Cuantos nodos desea:" << endl; |
|  | int n; |
|  | cin >> n; |
|  | head->val=1; |
|  | head->ante=NULL; |
|  | prev=head; |
|  | if (n>1){ |
|  | for (int i=2; i<=n; i++){ |
|  | cur=(struct node\*)malloc(sizeof(struct node)); |
|  | cur->val=i; |
|  | if (i==2){ |
|  | cur->ante=head; |
|  | } |
|  | vir=prev->next; |
|  | vir->ante=cur; |
|  | prev->next=cur; |
|  | prev=cur; |
|  | } |
|  | } |
|  | prev->next=NULL; |
|  | }; |
|  |  |
|  | void impr(struct node \*head) |
|  | { |
|  | cout << "IMPRIMIENDO:" << endl; |
|  | struct node \*t; |
|  | t=head; |
|  | while (t!=NULL){ |
|  | cout << t->val << endl; |
|  | t=t->next; |
|  | } |
|  | cout << "AHORA AL REVES:" << endl; |
|  | while (t!=NULL){ |
|  | cout << t->val << endl; |
|  | t=t->ante; |
|  | } |
|  | }; |
|  |  |
|  | struct node \*elim(struct node \*head) |
|  | { |
|  | cout << "Quitando nodo:" << endl << "Retire un termino existente:" << endl; |
|  | int key; |
|  | cin >> key; |
|  | if (head->val==key){ |
|  | struct node \*temp; |
|  | temp=head->next; |
|  | free(head); |
|  | head=temp; |
|  | return head; |
|  | } |
|  | else { |
|  | struct node \*prev, \*cur; |
|  | prev=head; |
|  | cur=head->next; |
|  | while(cur!=NULL){ |
|  | if (cur->val==key){ |
|  | prev->next=cur->next; |
|  | free(cur); |
|  | break; |
|  | } |
|  | prev=cur; |
|  | cur=cur->next; |
|  | } |
|  | return head; |
|  | } |
|  | }; |
|  |  |
|  | int main() |
|  | { |
|  | struct node \*head; |
|  | head=(struct node\*)malloc(sizeof(struct node)); |
|  | crear(head); |
|  | impr(head); |
|  | head=elim(head); |
|  | impr(head); |
|  | return 0; |
|  | } |

LISTA CIRCULAR:

|  |
| --- |
| #include <iostream> |
|  | #include <stdlib.h> |
|  |  |
|  | using namespace std; |
|  |  |
|  | struct node |
|  | { |
|  | int val; |
|  | struct node \*next; |
|  | }; |
|  |  |
|  | void crear(struct node \*head) |
|  | { |
|  | struct node \*prev, \*cur; |
|  | cout << "Cuantos nodos desea:" << endl; |
|  | int n; |
|  | cin >> n; |
|  | head->val=1; |
|  | prev=head; |
|  | if (n>1){ |
|  | for (int i=2; i<=n; i++){ |
|  | cur=(struct node\*)malloc(sizeof(struct node)); |
|  | cur->val=i; |
|  | prev->next=cur; |
|  | prev=cur; |
|  | } |
|  | } |
|  | prev->next=head; |
|  | }; |
|  |  |
|  | void impr(struct node \*head) |
|  | { |
|  | cout << "IMPRIMIENDO:" << endl; |
|  | struct node \*t; |
|  | t=head; |
|  | cout << t->val << endl; |
|  | t=t->next; |
|  | while (t!=head){ |
|  | cout << t->val << endl; |
|  | t=t->next; |
|  | } |
|  | }; |
|  |  |
|  | struct node \*elim(struct node \*head) |
|  | { |
|  | cout << "Quitando nodo:" << endl << "Retire un termino existente:" << endl; |
|  | int key; |
|  | cin >> key; |
|  | if (head->val==key){ |
|  | struct node \*temp; |
|  | temp=head->next; |
|  | free(head); |
|  | head=temp; |
|  | return head; |
|  | } |
|  | else { |
|  | struct node \*prev, \*cur; |
|  | prev=head; |
|  | cur=head->next; |
|  | while(cur!=NULL){ |
|  | if (cur->val==key){ |
|  | prev->next=cur->next; |
|  | free(cur); |
|  | break; |
|  | } |
|  | prev=cur; |
|  | cur=cur->next; |
|  | } |
|  | return head; |
|  | } |
|  | }; |
|  |  |
|  | int main() |
|  | { |
|  | struct node \*head; |
|  | head=(struct node\*)malloc(sizeof(struct node)); |
|  | crear(head); |
|  | impr(head); |
|  | head=elim(head); |
|  | impr(head); |
|  | return 0; |
|  | } |

LISTA DOBLE CIRCULAR:

|  |
| --- |
| #include <iostream> |
|  | #include <stdlib.h> |
|  |  |
|  | using namespace std; |
|  |  |
|  | struct node |
|  | { |
|  | int val; |
|  | struct node \*next, \*ante; |
|  | }; |
|  |  |
|  | void crear(struct node \*head) |
|  | { |
|  | struct node \*prev, \*cur, \*vir; |
|  | cout << "Cuantos nodos desea:" << endl; |
|  | int n; |
|  | cin >> n; |
|  | head->val=1; |
|  | prev=head; |
|  | if (n>1){ |
|  | for (int i=2; i<=n; i++){ |
|  | cur=(struct node\*)malloc(sizeof(struct node)); |
|  | cur->val=i; |
|  | if (i==2){ |
|  | cur->ante=head; |
|  | } |
|  | vir=prev->next; |
|  | vir->ante=cur; |
|  | prev->next=cur; |
|  | prev=cur; |
|  | } |
|  | } |
|  | prev->next=head; |
|  | head->ante=prev; |
|  | }; |
|  |  |
|  | void impr(struct node \*head) |
|  | { |
|  | cout << "IMPRIMIENDO:" << endl; |
|  | struct node \*t; |
|  | t=head; |
|  | cout << t->val << endl; |
|  | t=t->next; |
|  | while (t!=head){ |
|  | cout << t->val << endl; |
|  | t=t->next; |
|  | } |
|  | cout << "AHORA AL REVES:" << endl; |
|  | while (t!=head){ |
|  | cout << t->val << endl; |
|  | t=t->ante; |
|  | } |
|  | cout << head->val << endl; |
|  | }; |
|  |  |
|  | struct node \*elim(struct node \*head) |
|  | { |
|  | cout << "Quitando nodo:" << endl << "Retire un termino existente:" << endl; |
|  | int key; |
|  | cin >> key; |
|  | if (head->val==key){ |
|  | struct node \*temp; |
|  | temp=head->next; |
|  | free(head); |
|  | head=temp; |
|  | return head; |
|  | } |
|  | else { |
|  | struct node \*prev, \*cur; |
|  | prev=head; |
|  | cur=head->next; |
|  | while(cur!=NULL){ |
|  | if (cur->val==key){ |
|  | prev->next=cur->next; |
|  | free(cur); |
|  | break; |
|  | } |
|  | prev=cur; |
|  | cur=cur->next; |
|  | } |
|  | return head; |
|  | } |
|  | }; |
|  |  |
|  | int main() |
|  | { |
|  | struct node \*head; |
|  | head=(struct node\*)malloc(sizeof(struct node)); |
|  | crear(head); |
|  | impr(head); |
|  | head=elim(head); |
|  | impr(head); |
|  | return 0; |
|  | } |

PILA:

|  |
| --- |
| #include <iostream> |
|  | #include <windows.h> |
|  | using namespace std; |
|  |  |
|  | struct nodo{ |
|  | int nro; |
|  | struct nodo \*sgte; |
|  | }; |
|  |  |
|  | typedef nodo \*ptrPila; // creando nodo tipo puntero( tipo de dato ) |
|  |  |
|  | /\* Apilar elemento |
|  | ------------------------------------------------------------------------\*/ |
|  | void push( ptrPila &p, int valor ) |
|  | { |
|  | ptrPila aux; |
|  | aux = new(struct nodo); // apuntamos al nuevo nodo creado |
|  | aux->nro = valor; |
|  |  |
|  | aux->sgte = p ; |
|  | p = aux ; |
|  | } |
|  |  |
|  | /\* Desapilar elemento(devuelve elemento) |
|  | ------------------------------------------------------------------------\*/ |
|  | int pop( ptrPila &p ) |
|  | { |
|  | int num ; |
|  | ptrPila aux; |
|  |  |
|  | aux = p ; |
|  | num = aux->nro; // asignamos el primer vamor de la pila |
|  |  |
|  | p = aux->sgte ; |
|  | delete(aux); |
|  |  |
|  | return num; |
|  | } |
|  |  |
|  | /\* Muestra elementos de la pila |
|  | ------------------------------------------------------------------------\*/ |
|  | void mostrar\_pila( ptrPila p ) |
|  | { |
|  | ptrPila aux; |
|  | aux = p; // apunta al inicio de la lista |
|  |  |
|  | while( aux !=NULL ) |
|  | { |
|  | cout<<"\t"<< aux->nro <<endl; |
|  | aux = aux->sgte; |
|  | } |
|  | } |
|  |  |
|  | /\* Eliminar todos los elementos de la pila |
|  | ------------------------------------------------------------------------\*/ |
|  | void destruir\_pila( ptrPila &p) |
|  | { |
|  | ptrPila aux; |
|  |  |
|  | while( p != NULL) |
|  | { |
|  | aux = p; |
|  | p = aux->sgte; |
|  | delete(aux); |
|  | } |
|  | } |
|  |  |
|  | /\* Menu de opciones |
|  | ------------------------------------------------------------------------\*/ |
|  | void menu() |
|  | { |
|  | cout<<"\n\t CREANDO PILA, ¿QUÉ ESEA HACER?\n\n"; |
|  | cout<<" 1. APILAR "<<endl; |
|  | cout<<" 2. DESAPILAR "<<endl; |
|  | cout<<" 3. MOSTRAR PILA "<<endl; |
|  | cout<<" 4. DESTRUIR PILA "<<endl; |
|  | cout<<" 5. SALIR "<<endl; |
|  |  |
|  | cout<<"\n INGRESE OPCION: "; |
|  | } |
|  |  |
|  | /\* Funcion Principal |
|  | ------------------------------------------------------------------------\*/ |
|  | int main() |
|  | { |
|  | ptrPila p = NULL; // creando pila |
|  | int dato; |
|  | int op; |
|  | int x ; // numero que devuelve la funcon pop |
|  |  |
|  | system("color 0b"); |
|  |  |
|  | do |
|  | { |
|  | menu(); cin>> op; |
|  |  |
|  | switch(op) |
|  | { |
|  | case 1: |
|  |  |
|  | cout<< "\n NUMERO A APILAR: "; cin>> dato; |
|  | push( p, dato ); |
|  | cout<<"\n\n\t\tNumero " << dato << " apilado...\n\n"; |
|  | break; |
|  |  |
|  |  |
|  | case 2: |
|  |  |
|  | x = pop( p ); |
|  | cout<<"\n\n\t\tNumero "<< x <<" desapilado...\n\n"; |
|  | break; |
|  |  |
|  |  |
|  | case 3: |
|  |  |
|  | cout << "\n\n MOSTRANDO PILA\n\n"; |
|  | if(p!=NULL) |
|  | mostrar\_pila( p ); |
|  | else |
|  | cout<<"\n\n\tPila vacia..!"<<endl; |
|  | break; |
|  |  |
|  |  |
|  | case 4: |
|  |  |
|  | destruir\_pila( p ); |
|  | cout<<"\n\n\t\tPila eliminada...\n\n"; |
|  | break; |
|  |  |
|  | } |
|  |  |
|  | cout<<endl<<endl; |
|  | system("pause"); system("cls"); |
|  |  |
|  | }while(op!=5); |
|  |  |
|  |  |
|  | return 0; |
|  | } |

ARBOL BINARIO:

|  |
| --- |
| #include <iostream> |
|  | #include <cstdlib> |
|  | using namespace std; |
|  |  |
|  | struct nodo{ |
|  | int nro; |
|  | struct nodo \*izq, \*der; |
|  | }; |
|  |  |
|  | typedef struct nodo \*ABB; |
|  | /\* es un puntero de tipo nodo que hemos llamado ABB, que ulitizaremos |
|  | para mayor facilidad de creacion de variables \*/ |
|  |  |
|  | ABB crearNodo(int x) |
|  | { |
|  | ABB nuevoNodo = new(struct nodo); |
|  | nuevoNodo->nro = x; |
|  | nuevoNodo->izq = NULL; |
|  | nuevoNodo->der = NULL; |
|  |  |
|  | return nuevoNodo; |
|  | } |
|  | void insertar(ABB &arbol, int x) |
|  | { |
|  | if(arbol==NULL) |
|  | { |
|  | arbol = crearNodo(x); |
|  | } |
|  | else if(x < arbol->nro) |
|  | insertar(arbol->izq, x); |
|  | else if(x > arbol->nro) |
|  | insertar(arbol->der, x); |
|  | } |
|  |  |
|  | void preOrden(ABB arbol) |
|  | { |
|  | if(arbol!=NULL) |
|  | { |
|  | cout << arbol->nro <<" "; |
|  | preOrden(arbol->izq); |
|  | preOrden(arbol->der); |
|  | } |
|  | } |
|  |  |
|  | void enOrden(ABB arbol) |
|  | { |
|  | if(arbol!=NULL) |
|  | { |
|  | enOrden(arbol->izq); |
|  | cout << arbol->nro << " "; |
|  | enOrden(arbol->der); |
|  | } |
|  | } |
|  |  |
|  | void postOrden(ABB arbol) |
|  | { |
|  | if(arbol!=NULL) |
|  | { |
|  | postOrden(arbol->izq); |
|  | postOrden(arbol->der); |
|  | cout << arbol->nro << " "; |
|  | } |
|  | } |
|  |  |
|  | void verArbol(ABB arbol, int n) |
|  | { |
|  | if(arbol==NULL) |
|  | return; |
|  | verArbol(arbol->der, n+1); |
|  |  |
|  | for(int i=0; i<n; i++) |
|  | cout<<" "; |
|  |  |
|  | cout<< arbol->nro <<endl; |
|  |  |
|  | verArbol(arbol->izq, n+1); |
|  | } |
|  |  |
|  | int main() |
|  | { |
|  | ABB arbol = NULL; // creado Arbol |
|  |  |
|  | int n; // numero de nodos del arbol |
|  | int x; // elemento a insertar en cada nodo |
|  |  |
|  | cout << "\n\t\t ..[ ARBOL BINARIO DE BUSQUEDA ].. \n\n"; |
|  |  |
|  | cout << " Numero de nodos del arbol: "; |
|  | cin >> n; |
|  | cout << endl; |
|  |  |
|  | for(int i=0; i<n; i++) |
|  | { |
|  | cout << " Numero del nodo " << i+1 << ": "; |
|  | cin >> x; |
|  | insertar( arbol, x); |
|  | } |
|  |  |
|  | cout << "\n Mostrando ABB \n\n"; |
|  | verArbol( arbol, 0); |
|  |  |
|  | cout << "\n Recorridos del ABB"; |
|  |  |
|  | cout << "\n\n En orden : "; enOrden(arbol); |
|  | cout << "\n\n Pre Orden : "; preOrden(arbol); |
|  | cout << "\n\n Post Orden : "; postOrden(arbol); |
|  |  |
|  | cout << endl << endl; |
|  |  |
|  | system("pause"); |
|  | return 0; |
|  | } |