UNSA

UNIVERSIDAD NACIONAL DE SAN AGUSTÍN DE AREQUIPA

“UNIVERSIDAD NACIONAL DE SAN AGUSTÍN”

FACULTAD DE INGENIERÍA,PRODUCCIÓN Y SERVICIOS
ESCUELA PROFESIONAL DE CIENCIA DE LA COMPUTACIÓN

CURSO:

Ciencias de la Computación - Grupo “B”

DOCENTE:

Enzo Edir Velásquez Lobatón

ALUMNO:

Fabricio Huaquisto Quispe

REPOSITORIO:

https://github.com/fhuaquisto21/EPCC-CCII

Arequipa - Perú

2022

PILAS

1. **node.h**

```
class Node {
  private:
    int value;
    Node* next;
  public:
    Node(int);
    ~Node();
    int getValue();
    Node* getNext();
    void setValue(int);
    void setNext(Node*);
};
```

2. **node.cpp**

```
#include <iostream>
#include "node.h"

Node::Node(int _value) {
  this->value = _value;
  this->next = nullptr;
}

Node::~Node() {}

Node* Node::getNext() {
  return this->next;
}

int Node::getValue() {
  return this->value;
}

void Node::setNext(Node* _next) {
  this->next = _next;
}

void Node::setValue(int _value) {
  this->value = _value;
}
```

3. **pila.h**

```
#include "node.cpp"

class Pila {
  private:
    Node* head;
    int length;
  public:
    Pila();
    Pila(int);
    ~Pila();
    int push(int);
    int pop();
    void printPila();
    int search(int);
};
```

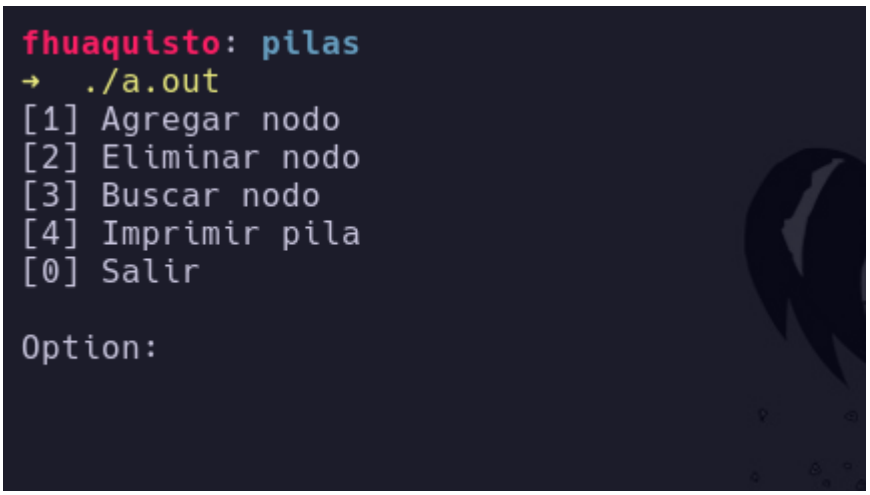
4. **pila.cpp**

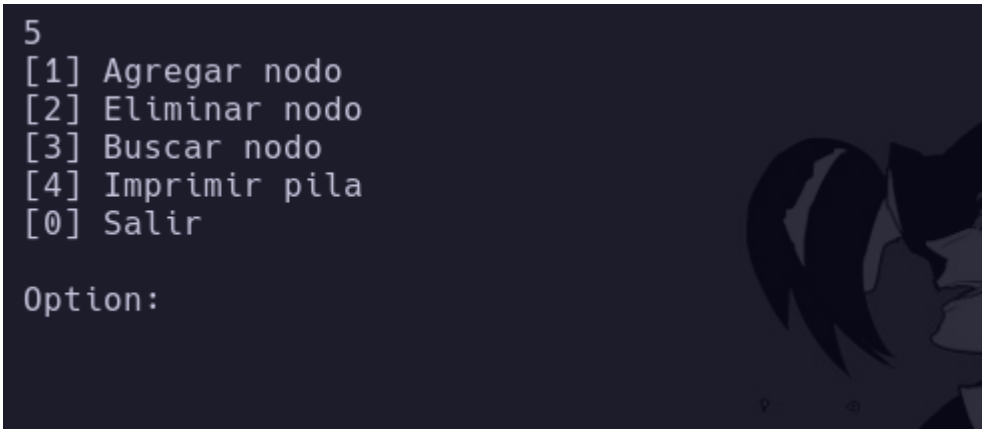
```
#include "pila.h"

Pila::Pila() {
  this->head = nullptr;
  this->length = 0;
}

Pila::Pila(int _value) {
  this->head = new Node(_value);
  this->length = 1;
}

Pila::~Pila() {}

int Pila::push(int _value) {
  Node* newNode = new Node(_value);
  if (this->head != nullptr) {
    newNode->setNext(this->head);
  }
  this->head = newNode;
  ++this->length;
  return this->head->getValue();
}

int Pila::pop() {
  if (this->head != nullptr) {
    Node* auxNode = this->head;
    int auxNodeValue = auxNode->getValue();
    this->head = this->head->getNext();
    delete auxNode;
    --this->length;
    return auxNodeValue;
  }
  return 0;
}

void Pila::printPila() {
  if (this->head == nullptr) {
    std::cerr << "ERROR: La pila está vacía";
    exit(-1);
  }
  Node* currentNode = this->head;
  while (currentNode->getNext() != nullptr) {
    std::cout << currentNode->getValue() << " -> ";
    currentNode = currentNode->getNext();
```

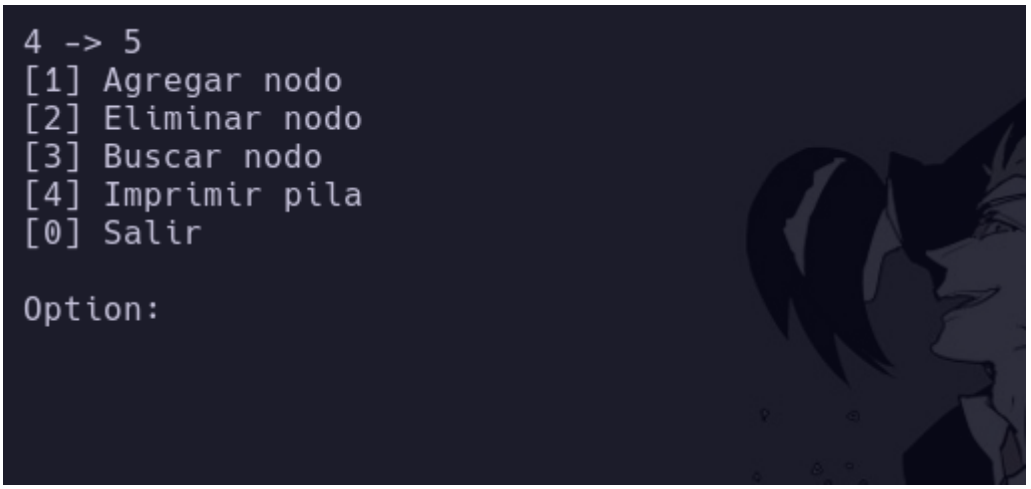
```
    }
    std::cout << currentNode->getValue() << std::endl;
}

int Pila::search(int _i) {
    if (_i >= this->length || _i < 0) {
        return 0;
    }
    Node* currentNode = this->head;
    for (int i = 0; i < _i; ++i) {
        currentNode = currentNode->getNext();
    }
    return currentNode->getValue();
}
```

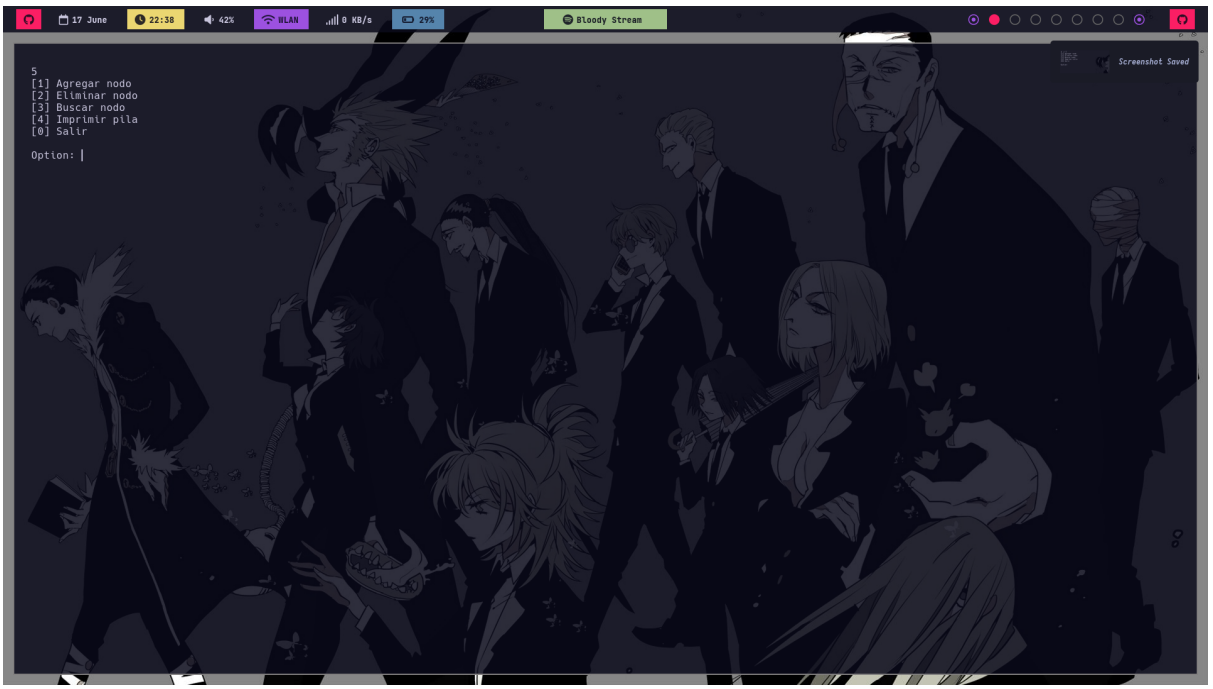
5. **main.cpp**

```
#include <iostream>
#include "pila.cpp"
using namespace std;

void printMenu() {
  cout << "[1] Agregar nodo" << endl;
  cout << "[2] Eliminar nodo" << endl;
  cout << "[3] Buscar nodo" << endl;
  cout << "[4] Imprimir pila" << endl;
  cout << "[0] Salir" << endl;
  cout << endl << "Option: ";
}

int main() {
  Pila* pila = new Pila();
  int opt = 0;
  int value;
  do {
    printMenu();
    cin >> opt;
    printf("\e[1;1H\e[2J");
    switch (opt) {
      case 0:
        break;
      case 1:
        cout << "Valor del nuevo nodo: ";
        cin >> value;
        pila->push(value);
        break;
      case 2:
        pila->pop();
        break;
      case 3:
        cout << "ïndice del nodo a buscar: ";
        cin >> value;
        cout << "Su valor es: " << pila->search(value) << endl;
        break;
      case 4:
        pila->printPila();
        break;
    }
  } while (opt != 0);
}
```


fhuaquisto: pilas
→ ./a.out
[1] Agregar nodo
[2] Eliminar nodo
[3] Buscar nodo
[4] Imprimir pila
[0] Salir
Option:


5
[1] Agregar nodo
[2] Eliminar nodo
[3] Buscar nodo
[4] Imprimir pila
[0] Salir
Option:


4 -> 5
[1] Agregar nodo
[2] Eliminar nodo
[3] Buscar nodo
[4] Imprimir pila
[0] Salir
Option:




```
5
[1] Agregar nodo
[2] Eliminar nodo
[3] Buscar nodo
[4] Imprimir pila
[0] Salir

Option:
```

```
ïndice del nodo a buscar: 0
Su valor es: 5
[1] Agregar nodo
[2] Eliminar nodo
[3] Buscar nodo
[4] Imprimir pila
[0] Salir

Option:
```

HANOI

1. **node.h**

```
class Node {
  private:
    int value;
    Node* next;
  public:
    Node(int);
    ~Node();
    int getValue();
    Node* getNext();
    void setValue(int);
    void setNext(Node*);
};
```

2. **node.cpp**

```
#include <iostream>
#include "node.h"

Node::Node(int _value) {
  this->value = _value;
  this->next = nullptr;
}

Node::~Node() {}

Node* Node::getNext() {
  return this->next;
}

int Node::getValue() {
  return this->value;
}

void Node::setNext(Node* _next) {
  this->next = _next;
}

void Node::setValue(int _value) {
  this->value = _value;
}
```

3. tower.h

```
#include "node.cpp"

class Tower {
  private:
    Node* head;
    void addNode(int);
  public:
    Tower(int);
    ~Tower();
    Node* getHead();
    int push(int);
    int pop();
    void printTower();
};
```

4. **tower.cpp**

```
#include <iostream>
#include "tower.h"

Tower::Tower(int _level) {
  this->head = new Node(0);
  for (int i = 0; i < _level - 1; ++i) {
    this->addNode(0);
  }
}

Tower::~Tower() {}

Node* Tower::getHead() {
  return this->head;
}

void Tower::addNode(int _value) {
  Node* newNode = new Node(_value);
  if (this->head == nullptr) {
    this->head = newNode;
  }
  Node* auxNode = this->head;
  newNode->setNext(auxNode);
  this->head = newNode;
  this->head->setNext(auxNode);
}

int Tower::push(int _value) {
  Node* currentNode = this->head;
  while (currentNode->getNext() != nullptr) {
    if (currentNode->getNext()->getValue() != 0) {
      currentNode->setValue(_value);
      break;
    }
    currentNode = currentNode->getNext();
  }
  currentNode->setValue(_value);
  return _value;
}

int Tower::pop() {
  Node* currentNode = this->head;
  int auxValue = 0;
  while (currentNode != nullptr) {
    if (currentNode->getValue() != 0) {
      auxValue = currentNode->getValue();
      currentNode->setValue(0);
```

```
            break;
        }
        currentNode = currentNode->getNext();
    }
    return auxValue;
}

void Tower::printTower() {
    Node* auxNode = this->head;
    while (auxNode->getNext() != nullptr) {
        std::cout << auxNode->getValue() << " -> ";
        auxNode = auxNode->getNext();
    }
    std::cout << auxNode->getValue() << std::endl;
}
```

5. hanoi.h

```
#include "tower.cpp"

class Hanoi {
  private:
    Tower** towers;
    int level;
    int** transformHanoi();
  public:
    Hanoi(int);
    ~Hanoi();
    void resolveGame(int, Tower*, Tower*, Tower*);
    void printMove();
    Tower** getTower();
};
```

6. hanoi.cpp

```
#include <iostream>
#include "hanoi.h"

Hanoi::Hanoi(int _level) {
  this->level = _level;
  this->towers = new Tower* [3];
  for (int i = 0; i < 3; ++i) {
    this->towers[i] = new Tower(_level);
  }
  for (int i = _level; i > 0; --i) {
    this->towers[0]->push(i);
  }
}

Hanoi::~Hanoi() {}

void drawRow(int _value, int _maxLevel) {
  for (int i = 0; i < _maxLevel - (_value == 0 ? 1 : _value); ++i) {
    std::cout << " ";
  }
  if (_value != 0) {
    for (int i = 0; i < (_value * 2) - 1; ++i) {
      std::cout << "#";
    }
  } else {
    std::cout << "|";
  }
  for (int i = 0; i < _maxLevel - (_value == 0 ? 1 : _value); ++i) {
    std::cout << " ";
  }
```

```
}

int** Hanoi::transformHanoi() {
  int** values = new int* [3];
  for (int i = 0; i < 3; ++i) {
    int* towerValues = new int[this->level];
    Tower* auxTower = this->towers[i];
    Node* auxNode = auxTower->getHead();
    for (int j = 0; j < this->level; ++j) {
      towerValues[j] = auxNode->getValue();
      auxNode = auxNode->getNext();
    }
    values[i] = towerValues;
  }
  return values;
}

void Hanoi::printMove() {
  int** values = this->transformHanoi();
  for (int i = 0; i <= this->level; ++i) {
    for (int j = 0; j < 3; ++j) {
      if (i == 0) {
        drawRow(0, this->level);
        std::cout << "\t";
      } else {
        drawRow(values[j][i - 1], this->level);
        std::cout << "\t";
      }
    }
    std::cout << std::endl;
  }
  std::cout << std::endl << std::endl;
}

void auxResolver(int _n, Tower* _a, Tower* _b, Tower* _c) {
  if (_n == 1) {
    _a->push(_c->pop());
  } else {
    auxResolver(_n - 1, _a, _b, _c);
    _a->push(_c->pop());
    auxResolver(_n - 1, _b, _c, _a);
  }
}

void Hanoi::resolveGame(int aux, Tower* A, Tower* C, Tower* B) {
  if (aux == 1) {
    C->push(A->pop());
    this->printMove();
```

```
  } else {
    this->resolveGame(aux - 1, A, B, C);
    C->push(A->pop());
    this->printMove();
    this->resolveGame(aux - 1, B, C, A);
  }
}

Tower** Hanoi::getTower() {
  return this->towers;
}
```

7. main.cpp

```
#include <iostream>
#include "hanoi.cpp"

int main() {
  int level;
  std::cout << "Cantidad de fichas: ";
  std::cin >> level;
  Hanoi* hanoi = new Hanoi(level);
  hanoi->printMove();
  hanoi->resolveGame(level, hanoi->getTower()[0], hanoi->getTower()[2],
hanoi->getTower()[1]);
  return 0;
}
```