Tipo de Dados Abstratos

1. Ponto.hpp

```
#pragma once
#include <iostream>
using namespace std;

class Ponto {
        public:
                Ponto(double x, double y);
                ~Ponto();
                double getX()  const ;
                double getY()  const ;
                void setX(double x);
                void setY(double y);
                double distancia(Ponto & p);
                Ponto operator+(const Ponto & p);
                friend ostream &operator<<( ostream
                ↪  &output, const Ponto &p );
        private:
                double x, y;
};
```

2. Ponto.cpp

```
#include "Ponto.hpp"
#include <cmath>


using namespace std;

Ponto::Ponto(double x, double y){
        this->x = x;
        this->y = y;
        cout << *this << "criado" << endl;
}

Ponto::~Ponto(){
        cout << *this << "destruido" << endl;
}
```

```
double Ponto::getX()  const { return x; }

double Ponto::getY()  const { return y; }

void Ponto::setX(double x) { this->x = x; }

void Ponto::setY(double y) { this->y = y; }

double Ponto::distancia(Ponto & p) {
        double dx = x - p.getX();
        double dy = y - p.getY();
        return sqrt(dx*dx + dy*dy);
}

//Sobrecarga do operador +
Ponto Ponto::operator+(const Ponto & p){
        return Ponto(this->x + p.getX() , this->y +
        ↪  p.getY());
}

//Sobrecarga do operador <<
ostream & operator<<( ostream &output, const Ponto &p ){
        output << "Ponto(" << p.getX() << "," << p.getY()
        ↪  << ")";
        return output;
}
```

3. Círculo.hpp

```
#pragma once
#include "Ponto.hpp"
#include <iostream>
using namespace std;

class Circulo {
        public:
                Circulo(Ponto centro, double raio);
                ~Circulo();
                double getRaio() const;
                void setRaio(double raio);
                Ponto getCentro() const;
```

```
                bool interior(Ponto & p);
                double area();
                friend ostream &operator<<( ostream
                ↪  &output, const Circulo &circ );
        private:
                Ponto centro;
                double raio;
};
```

4. Circulo.cpp

```
#include "Circulo.hpp"
#include <cmath>

using namespace std;

Circulo::Circulo(Ponto centro, double raio) :
↪  centro(centro), raio(raio) {
        cout << *this << "construido" << endl;
}

Circulo::~Circulo(){
        cout << *this << "destruido" << endl;
}

double Circulo::getRaio() const{
        return this->raio;
}

void Circulo::setRaio(double raio){
        this->raio = raio;
}

Ponto Circulo::getCentro() const{
        return this->centro;
}

bool Circulo::interior(Ponto & p){
        return p.distancia(this->centro) <= this->raio;
}
```

```
double Circulo::area(){
        return M_PI*this->raio*this->raio;
}

ostream &operator<<( ostream &output, const Circulo &circ
↪  ){
        output << "Circulo( centro = " <<
        ↪  circ.getCentro() << " , raio = " <<
        ↪  circ.getRaio() << ")";
        return output;
}
```

5. Conjunto.hpp

```
#pragma once
#include <iostream>
#include <vector>
using namespace std;

//template <typename T> class Conjunto;

template <typename T>
class Conjunto {
        public:
                Conjunto(){
                }
                ~Conjunto(){
                }

                void insert(T value){
                        if(!find(value)){
                                int j = arr.size()-1;
                                arr.push_back(value);
                                while( j >= 0 && arr[j] >
                                ↪  value){
                                        arr[j+1] =
                                        ↪  arr[j];
                                        j--;
                                }
                                arr[j+1] = value;
```

```
        }
}
void remove(T value){
        int i;

        for(i = 0; i < arr.size(); i++){
                if(arr[i] == value)
                ↪  break;
        }

        for(int j = i; j < arr.size()-1;
        ↪  j++){
                arr[j] = arr[j+1];
        }

        arr.pop_back();
}
bool find(T value){
        int start = 0;
        int end = arr.size() - 1;
        while( start < end ){
                int mid = (start+end)/2;
                if(arr[mid] == value){
                        return true;
                }else if( arr[mid] >
                ↪  value){
                        end = mid-1;
                }else{
                        start = mid+1;
                }
        }
        return false;
}
int size() const{
        return arr.size();
}
friend ostream &operator<<( ostream
↪  &output, const Conjunto<T> &c ){
        output << "[";
        output << c.arr[0];
```

```
                        for(int i = 1; i < c.size() ;
                        ↪  i++)
                                output << "," <<
                                ↪  c.arr[i];
                        output << "]";
                        return output;
                }
        private:
                vector <T> arr;
};
```

6. Josephus.hpp

```
#pragma once
class Josephus {
    private:
        int n, e;
        // vetor usado para guardar os identificadores
        ↪  das pessoas
        int * elem;
        // vetor usado para checar se uma pessoa está
        ↪  viva
        bool * vivo;
        // método para descobrir a próxima pessoa viva
        int next(int pos);
        void kill(int pos);
    public:

        Josephus(int n, int e);
        int survivor();
};
#endif
```

7. Josephus.cpp

```
#include <iostream>
#include "Josephus.hpp"


using namespace std;

Josephus::Josephus(int n, int e) : n(n), e(e) {
```

```
    elem = new int[n];
    vivo = new bool[n];
    for(int i = 0; i < n; i++){
        elem[i] = i+1;
        vivo[i] = true;
    }
}


int Josephus::next(int pos){
    pos = (pos + 1)%n;
    while( !vivo[pos] ) pos = (pos+1)%n;
    return pos;
}

void Josephus::kill(int pos){
    vivo[pos] = false;
}
int Josephus::survivor(){
    int num_vivos = n;
    int pos = e-1;

    while( num_vivos > 1 ){
        pos = next(pos); // posicao do cara que vai
        ↪  morrer
        kill(pos);
        //cout << "morrendo" << elem[pos] << endl;
        pos = next(pos); // vai receber a espada
        num_vivos--;
    }
    return elem[pos];

}
```