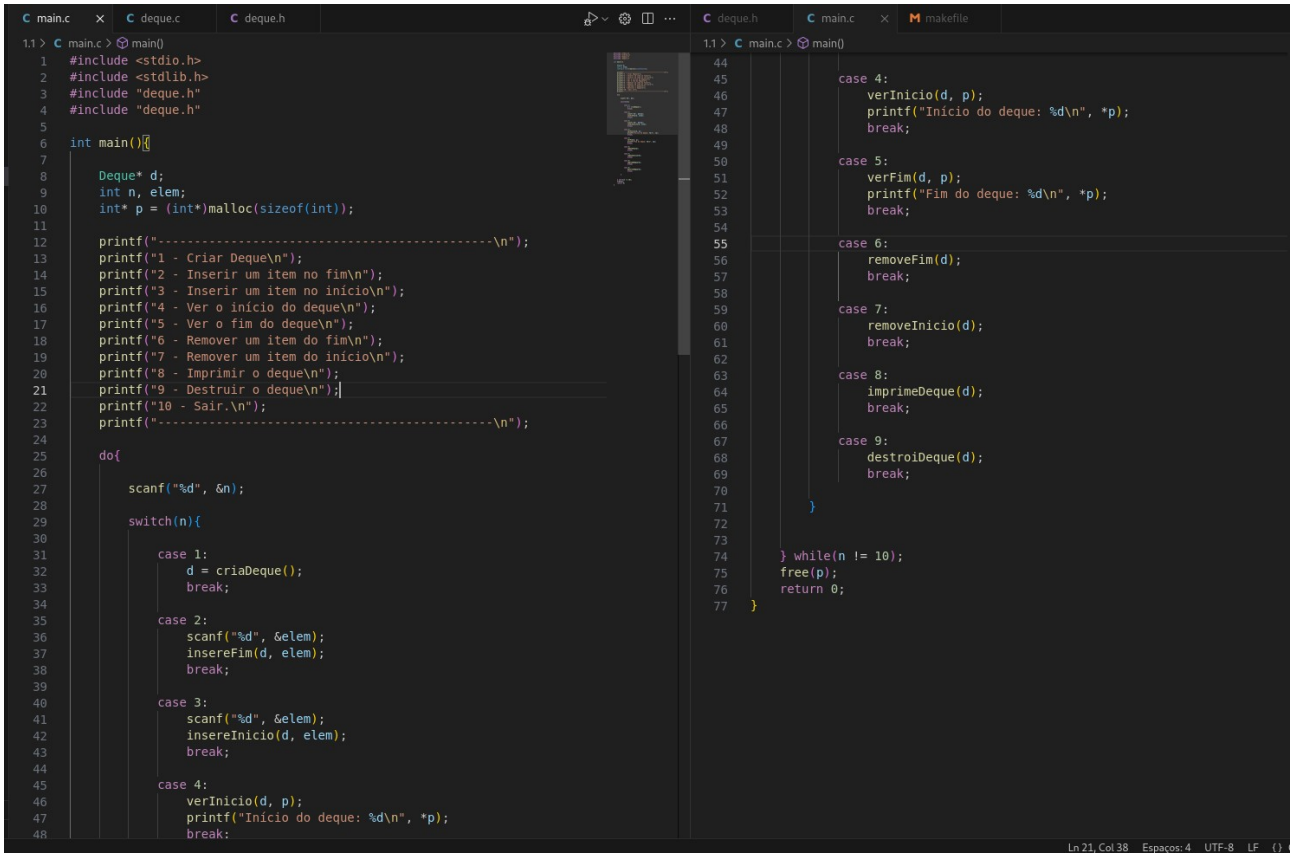
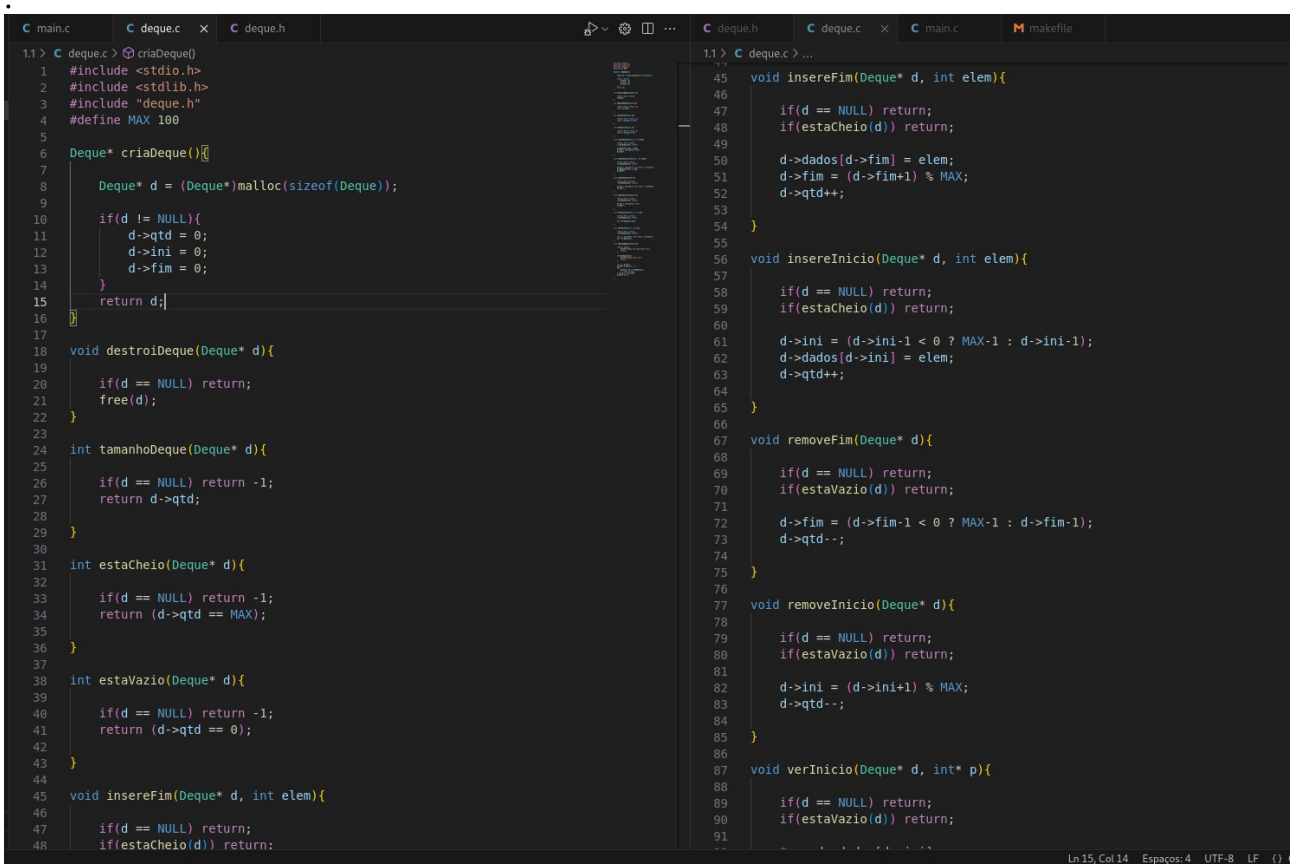
Roteiro 6 - LUCAS EDUARDO LEITE COSTA

1.1



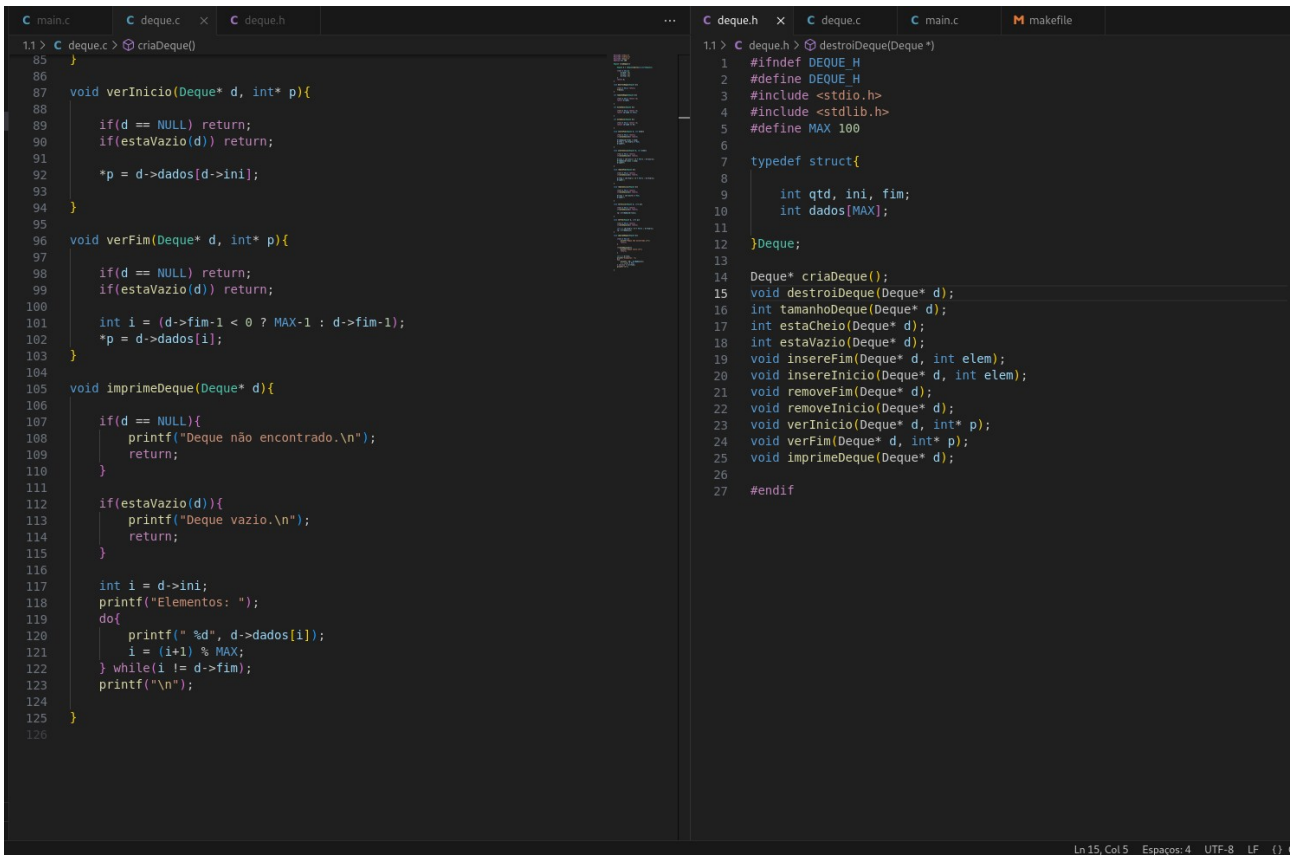
```c
#include <stdio.h>
#include <stdlib.h>
#include "deque.h"
#include "deque.h"

int main(){

    Deque* d;
    int n, elem;
    int* p = (int*)malloc(sizeof(int));

    printf("-----------------------------------------\n");
    printf("1 - Criar Deque\n");
    printf("2 - Inserir um item no fim\n");
    printf("3 - Inserir um item no inicio\n");
    printf("4 - Ver o início do deque\n");
    printf("5 - Ver o fim do deque\n");
    printf("6 - Remover um item do fim\n");
    printf("7 - Remover um item do início\n");
    printf("8 - Imprimir o deque\n");
    printf("9 - Destruir o deque\n");
    printf("10 - Sair.\n");
    printf("-----------------------------------------\n");

    do{

        scanf("%d", &n);

        switch(n){

            case 1:
                d = criaDeque();
                break;

            case 2:
                scanf("%d", &elem);
                insereFim(d, elem);
                break;

            case 3:
                scanf("%d", &elem);
                insereInicio(d, elem);
                break;

            case 4:
                verInicio(d, p);
                printf("Início do deque: %d\n", *p);
                break;

            case 5:
                verFim(d, p);
                printf("Fim do deque: %d\n", *p);
                break;

            case 6:
                removeFim(d);
                break;

            case 7:
                removeInicio(d);
                break;

            case 8:
                imprimeDeque(d);
                break;

            case 9:
                destroiDeque(d);
                break;

        }

    } while(n != 10);
    free(p);
    return 0;
}
```

.



```c
#include <stdio.h>
#include <stdlib.h>
#include "deque.h"
#define MAX 100

Deque* criaDeque(){

    Deque* d = (Deque*)malloc(sizeof(Deque));

    if(d != NULL){
        d->qtd = 0;
        d->ini = 0;
        d->fim = 0;
    }
    return d;
}

void destroiDeque(Deque* d){

    if(d == NULL) return;
    free(d);
}

int tamanhoDeque(Deque* d){

    if(d == NULL) return -1;
    return d->qtd;

}

int estaCheio(Deque* d){

    if(d == NULL) return -1;
    return (d->qtd == MAX);

}

int estaVazio(Deque* d){

    if(d == NULL) return -1;
    return (d->qtd == 0);

}

void insereFim(Deque* d, int elem){

    if(d == NULL) return;
    if(estaCheio(d)) return;

    d->dados[d->fim] = elem;
    d->fim = (d->fim+1) % MAX;
    d->qtd++;

}

void insereInicio(Deque* d, int elem){

    if(d == NULL) return;
    if(estaCheio(d)) return;

    d->ini = (d->ini-1 < 0 ? MAX-1 : d->ini-1);
    d->dados[d->ini] = elem;
    d->qtd++;

}

void removeFim(Deque* d){

    if(d == NULL) return;
    if(estaVazio(d)) return;

    d->fim = (d->fim-1 < 0 ? MAX-1 : d->fim-1);
    d->qtd--;

}

void removeInicio(Deque* d){

    if(d == NULL) return;
    if(estaVazio(d)) return;

    d->ini = (d->ini+1) % MAX;
    d->qtd--;

}

void verInicio(Deque* d, int* p){

    if(d == NULL) return;
    if(estaVazio(d)) return;
```

.



```
}

void verInicio(Deque* d, int* p){

    if(d == NULL) return;
    if(estaVazio(d)) return;

    *p = d->dados[d->ini];

}

void verFim(Deque* d, int* p){

    if(d == NULL) return;
    if(estaVazio(d)) return;

    int i = (d->fim-1 < 0 ? MAX-1 : d->fim-1);
    *p = d->dados[i];
}

void imprimeDeque(Deque* d){

    if(d == NULL){
        printf("Deque não encontrado.\n");
        return;
    }

    if(estaVazio(d)){
        printf("Deque vazio.\n");
        return;
    }

    int i = d->ini;
    printf("Elementos: ");
    do{
        printf(" %d", d->dados[i]);
        i = (i+1) % MAX;
    } while(i != d->fim);
    printf("\n");

}
```

```
#ifndef DEQUE_H
#define DEQUE_H
#include <stdio.h>
#include <stdlib.h>
#define MAX 100

typedef struct{

    int qtd, ini, fim;
    int dados[MAX];

}Deque;

Deque* criaDeque();
void destroiDeque(Deque* d);
int tamanhoDeque(Deque* d);
int estaCheio(Deque* d);
int estaVazio(Deque* d);
void insereFim(Deque* d, int elem);
void insereInicio(Deque* d, int elem);
void removeFim(Deque* d);
void removeInicio(Deque* d);
void verInicio(Deque* d, int* p);
void verFim(Deque* d, int* p);
void imprimeDeque(Deque* d);

#endif
```

```
[lucascosta@fedora 1.1]$ ./main
-------------------------------------------
1 - Criar Deque
2 - Inserir um item no fim
3 - Inserir um item no início
4 - Ver o início do deque
5 - Ver o fim do deque
6 - Remover um item do fim
7 - Remover um item do início
8 - Imprimir o deque
9 - Destruir o deque
10 - Sair.
-------------------------------------------
1
2
54
2
65
3
45
4
Início do deque: 45
5
Fim do deque: 65
6
7
8
Elementos:  54
8
Elementos:  54
9
10
[lucascosta@fedora 1.1]$
```

1.2

1.2 > C main.c > ...

```c
#include <stdio.h>
#include <stdlib.h>
#include "deque.h"

int main(){

    Deque* d;
    int n, elem;
    int* p = (int*)malloc(sizeof(int));

    printf("-----------------------------------------\n");
    printf("1 - Criar Deque\n");
    printf("2 - Inserir um item no fim\n");
    printf("3 - Inserir um item no inicio\n");
    printf("4 - Ver o inicio do deque\n");
    printf("5 - Ver o fim do deque\n");
    printf("6 - Remover um item do fim\n");
    printf("7 - Remover um item do inicio\n");
    printf("8 - Imprimir o deque\n");
    printf("9 - Destruir o deque\n");
    printf("10 - Sair.\n");
    printf("-----------------------------------------\n");

    do{

        scanf("%d", &n);

        switch(n){

            case 1:
                d = criaDeque();
                break;

            case 2:
                scanf("%d", &elem);
                insereFim(d, elem);
                break;

            case 3:
                scanf("%d", &elem);
                insereInicio(d, elem);
                break;

            case 4:
                verInicio(d, p);
                printf("Inicio do deque: %d\n", *p);
                break;

            case 4:
                verInicio(d, p);
                printf("Inicio do deque: %d\n", *p);
                break;

            case 5:
                verFim(d, p);
                printf("Fim do deque: %d\n", *p);
                break;

            case 6:
                removeFim(d);
                break;

            case 7:
                removeInicio(d);
                break;

            case 8:
                imprimeDeque(d);
                break;

            case 9:
                destroiDeque(d);
                break;

        }


    } while(n != 10);
    free(p);
    return 0;
}
```

.

1.2 > C deque.c > ...

```c
#include <stdio.h>
#include <stdlib.h>
#include "deque.h"

NO* alocarNO(){
    NO* n = (NO*)malloc(sizeof(NO));
    if(n != NULL){
        n->info = 0;
        n->prox = NULL;
        n->ant = NULL;
    }
    return n;
}

void destroiNO(NO* n){
    if(n == NULL) return;
    free(n);
}


Deque* criaDeque(){
    Deque* d = (Deque*)malloc(sizeof(Deque));
    if(d != NULL){
        d->qtd = 0;
        d->ini = NULL;
        d->fim = NULL;
    }

    return d;

}

void destroiDeque(Deque* d){
    if(d == NULL) return;

    NO* aux;
    while(d->ini != NULL){
        aux = d->ini;
        d->ini = d->ini->prox;
        free(aux);
    }
    free(d);

}

int tamanhoDeque(Deque* d){

    if(d == NULL) return -1;
```

1.2 > C deque.c > alocarNO()

```c
}

int tamanhoDeque(Deque* d){

    if(d == NULL) return -1;
    return d->qtd;

}

int estaVazio(Deque* d){

    if(d == NULL) return -1;
    return (d->qtd == 0);

}

void insereFim(Deque* d, int elem){
    if(d == NULL) return;

    NO* n = alocarNO();
    if(n == NULL) return;

    n->info = elem;
    n->prox = NULL;

    if(estaVazio(d)){
        n->ant = NULL;
        d->ini = n;
    }

    else{
        d->fim->prox = n;
        n->ant = d->fim;
    }

    d->fim = n;
    d->qtd++;

}

void insereInicio(Deque* d, int elem){

    if(d == NULL) return;

    NO* n = alocarNO();
    if(n == NULL) return;
```

.

1.2 > C deque.c > ...

```c

void insereInicio(Deque* d, int elem){

    if(d == NULL) return;

    NO* n = alocarNO();
    if(n == NULL) return;

    n->info = elem;
    n->ant = NULL;

    if(estaVazio(d)){
        n->prox = NULL;
        d->fim = n;
    }

    else{
        d->ini->ant = n;
        n->prox = d->ini;
    }

    d->ini = n;
    d->qtd++;

}

void removeFim(Deque* d){

    if(d == NULL) return;
    if(estaVazio(d)) return;
    NO* aux = d->fim;
    if(d->ini == d->fim){
        d->ini = NULL;
        d->fim = NULL;
    }
    else{
        d->fim = d->fim->ant;
        d->fim->prox = NULL;
    }
    destroiNO(aux);
    d->qtd--;
}

void removeInicio(Deque* d){

    if(d == NULL) return;
    if(estaVazio(d)) return;
```

1.2 > C deque.c > alocarNO()

```c

void removeInicio(Deque* d){

    if(d == NULL) return;
    if(estaVazio(d)) return;
    NO* aux = d->ini;
    if(d->ini == d->fim){
        d->ini = NULL;
        d->fim = NULL;
    }
    else{
        d->ini = d->ini->prox;
        d->ini->ant = NULL;
    }
    destroiNO(aux);
    d->qtd--;
}

void verInicio(Deque* d, int* p){

    if(d == NULL) return;
    if(estaVazio(d)) return;
    *p = d->ini->info;
}

void verFim(Deque* d, int* p){

    if(d == NULL) return;
    if(estaVazio(d)) return;
    *p = d->fim->info;

}

void imprimeDeque(Deque* d){

    if(d == NULL){
        printf("Deque não encontrado.\n");
        return;
    }

    if(estaVazio(d)){
        printf("Deque vazio.\n");
        return;
    }

    NO* aux = d->ini;

    printf("Elementos:");
```

•

1.2 > C deque.c > ...

```c
    *p = d->fim->info;

}

void imprimeDeque(Deque* d){

    if(d == NULL){
        printf("Deque não encontrado.\n");
        return;
    }

    if(estaVazio(d)){
        printf("Deque vazio.\n");
        return;
    }

    NO* aux = d->ini;

    printf("Elementos:");
    while(aux != d->fim->prox){
        printf(" %d", aux->info);
        aux = aux->prox;
    }
    printf("\n");

}
```

1.2 > C deque.h > __unnamed_struct_1f19_1 > fim

```c
#ifndef DEQUE_H
#define DEQUE_H
#include <stdio.h>
#include <stdlib.h>

typedef struct NO NO;
struct NO{
    int info;
    NO* prox;
    NO* ant;
};

typedef struct{
    int qtd;
    struct NO* ini;
    struct NO* fim;
}Deque;
NO* alocarNO();
void destroiNO(NO* n);
Deque* criaDeque();
void destroiDeque(Deque* d);
int tamanhoDeque(Deque* d);
int estaVazio(Deque* d);
void insereFim(Deque* d, int elem);
void insereInicio(Deque* d, int elem);
void removeFim(Deque* d);
void removeInicio(Deque* d);
void verInicio(Deque* d, int* p);
void verFim(Deque* d, int* p);
void imprimeDeque(Deque* d);

#endif
```

•

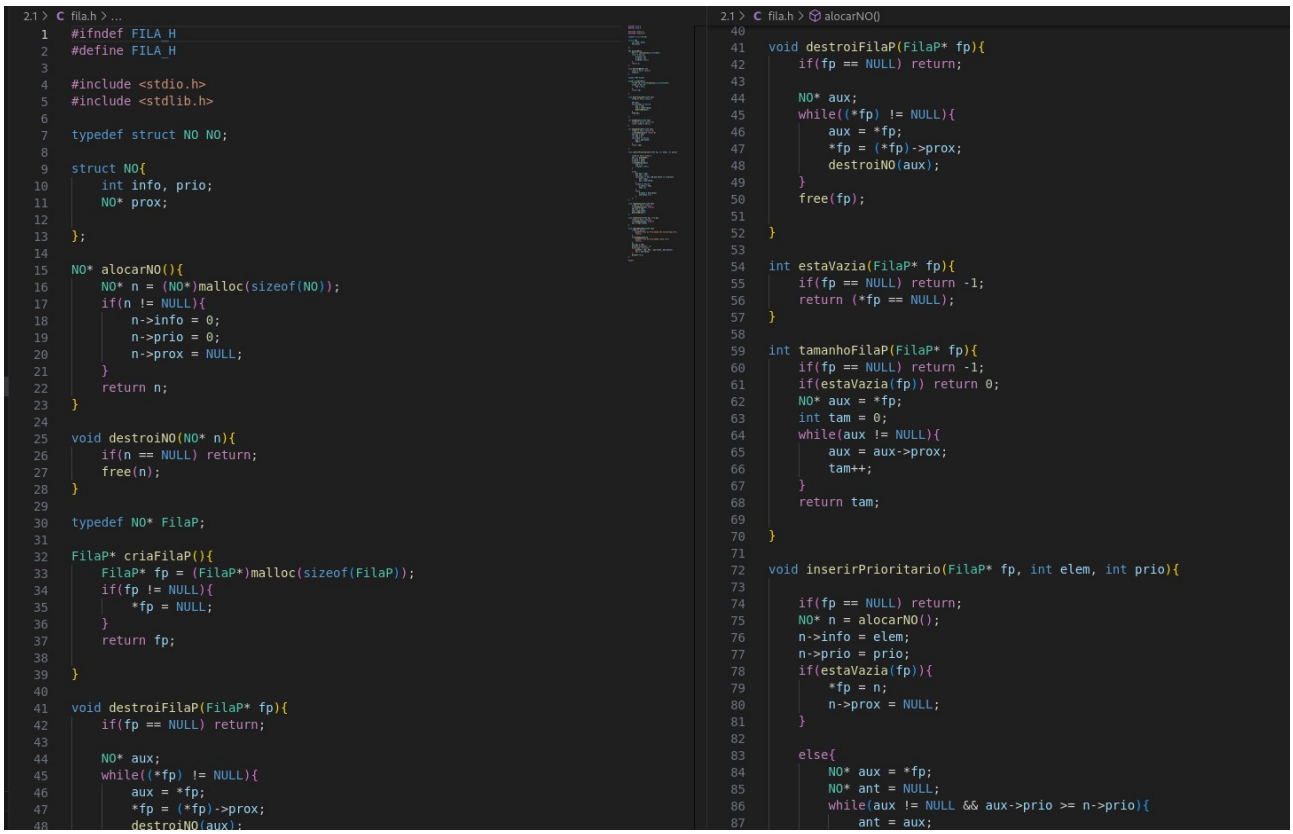
```
[lucascosta@fedora 1.2]$ ./main
-----------------------------------------------
1 - Criar Deque
2 - Inserir um item no fim
3 - Inserir um item no início
4 - Ver o início do deque
5 - Ver o fim do deque
6 - Remover um item do fim
7 - Remover um item do início
8 - Imprimir o deque
9 - Destruir o deque
10 - Sair.
-----------------------------------------------
1
2
58
2
96
3
45
4
Início do deque: 45
5
Fim do deque: 96
6
7
8
Elementos: 58
9
10
[lucascosta@fedora 1.2]$
```

2.1

2.1 > C main.c > main()

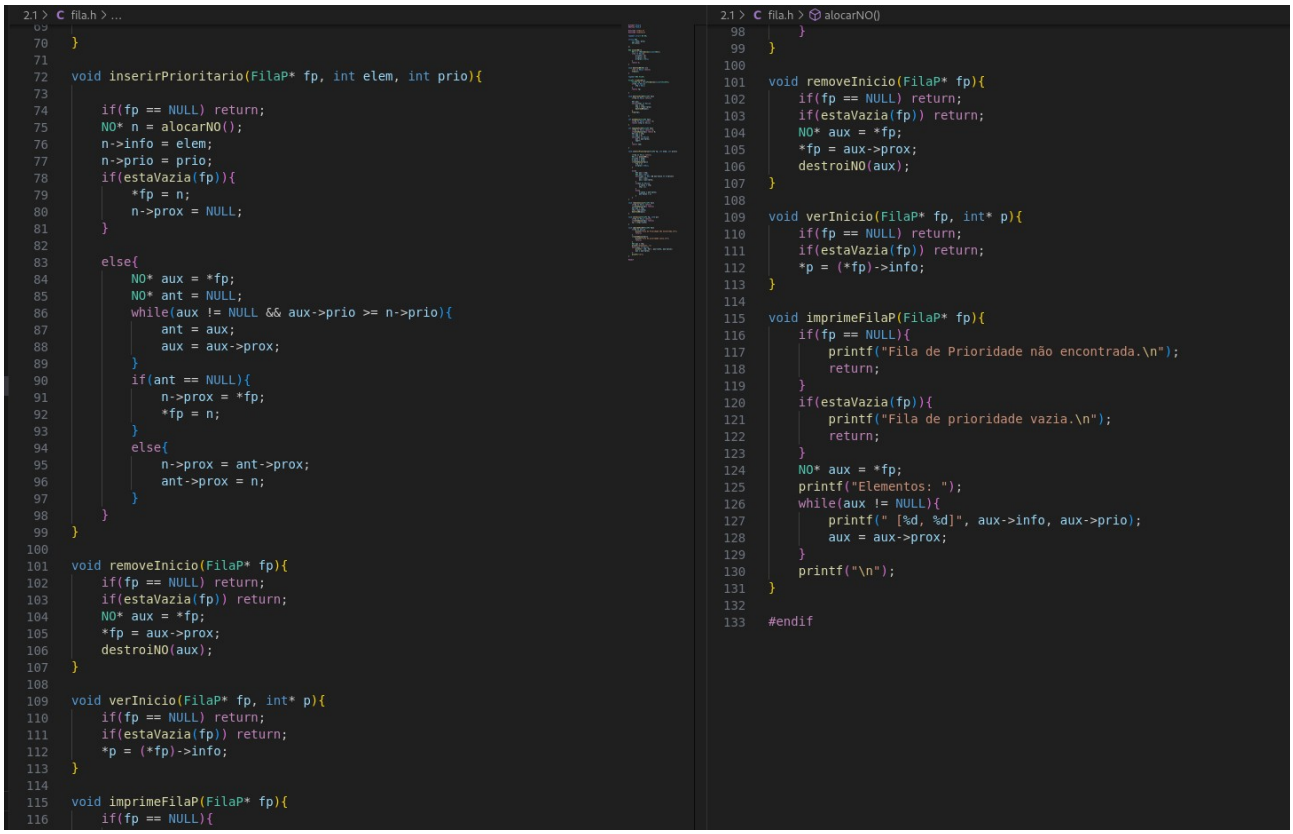
```c
#include <stdio.h>
#include <stdlib.h>
#include "fila.h"

int main(){

    FilaP* fp = NULL;
    int n, elem, prio;
    int* p = (int*)malloc(sizeof(int));

    printf("-----------------------------------------------\n");
    printf("Fila de Prioridade Simplesmente Encadeada:\n");
    printf("1 - Criar Fila;\n");
    printf("2 - Inserir um item pela prioridade;\n");
    printf("3 - Ver o início da Fila;\n");
    printf("4 - Remover um item;\n");
    printf("5 - Imprimir a Fila;\n");
    printf("6 - Mostrar o tamanho da Fila;\n");
    printf("7 - Destruir a Fila;\n");
    printf("8 - Sair.\n");
    printf("-----------------------------------------------\n");

    do{
        scanf("%d", &n);
        switch(n){
            case 1:
                fp = criaFilaP();
                break;
            case 2:
                printf("Elemento a ser inserido: ");
                scanf("%d", &elem);
                printf("Prioridade do elemento: ");
                scanf("%d", &prio);
                inserirPrioritario(fp, elem, prio);
                break;
            case 3:
                verInicio(fp, p);
                printf("Início da Fila: %d\n", *p);
                break;
            case 4:
                removeInicio(fp);
                break;
            case 5:
                imprimeFilaP(fp);
                break;
            case 6:
                printf("Tamanho da Fila: %d\n", tamanhoFilaP(fp));
                break;
            case 7:
                destroiFilaP(fp);
                break;
        }
    } while(n != 8);
    free(p);
    return 0;

}
```

.



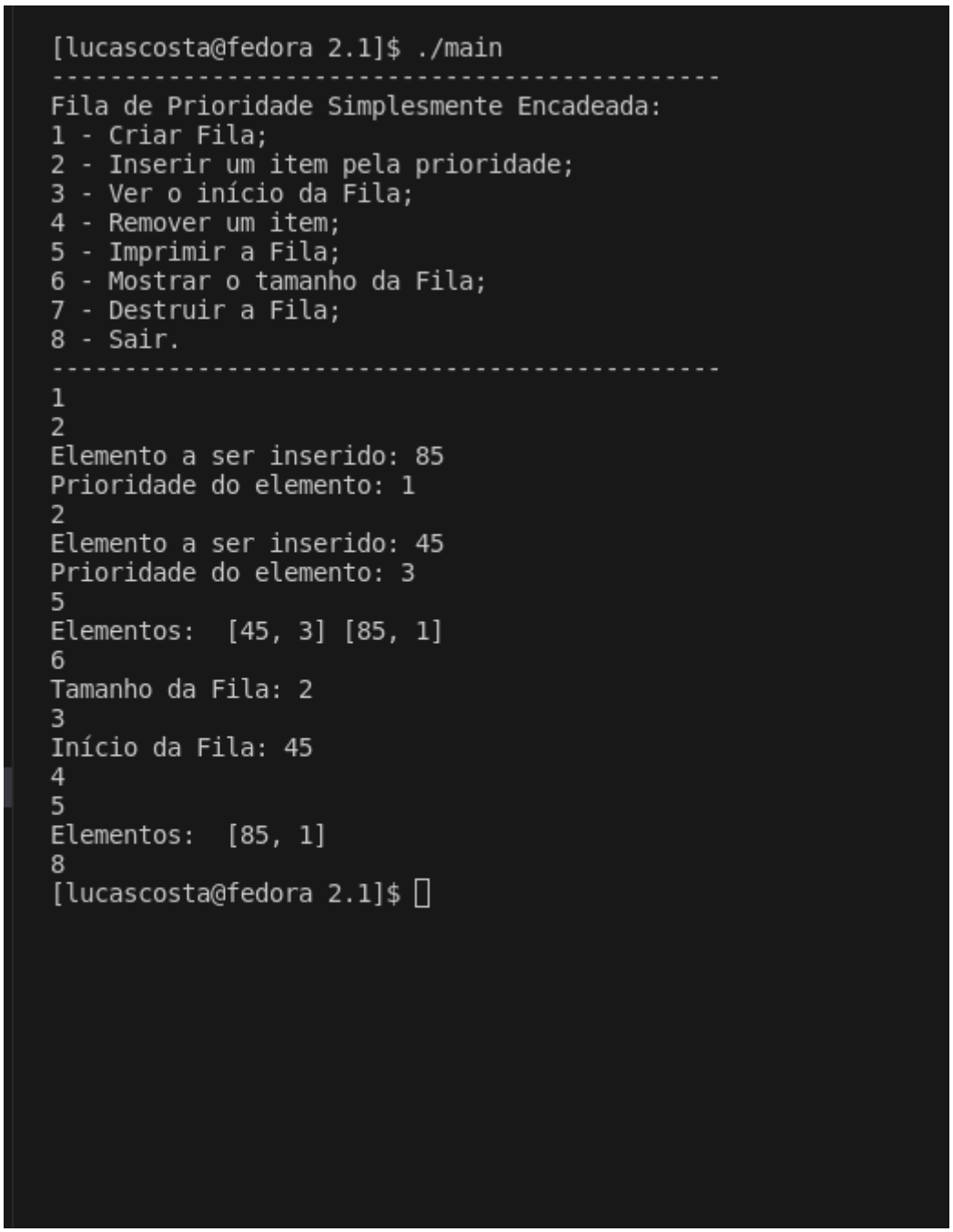
```c
#ifndef FILA_H
#define FILA_H

#include <stdio.h>
#include <stdlib.h>

typedef struct NO NO;

struct NO{
    int info, prio;
    NO* prox;

};

NO* alocarNO(){
    NO* n = (NO*)malloc(sizeof(NO));
    if(n != NULL){
        n->info = 0;
        n->prio = 0;
        n->prox = NULL;
    }
    return n;
}

void destroiNO(NO* n){
    if(n == NULL) return;
    free(n);
}

typedef NO* FilaP;

FilaP* criaFilaP(){
    FilaP* fp = (FilaP*)malloc(sizeof(FilaP));
    if(fp != NULL){
        *fp = NULL;
    }
    return fp;

}

void destroiFilaP(FilaP* fp){
    if(fp == NULL) return;

    NO* aux;
    while((*fp) != NULL){
        aux = *fp;
        *fp = (*fp)->prox;
        destroiNO(aux);
    }
    free(fp);

}

int estaVazia(FilaP* fp){
    if(fp == NULL) return -1;
    return (*fp == NULL);
}

int tamanhoFilaP(FilaP* fp){
    if(fp == NULL) return -1;
    if(estaVazia(fp)) return 0;
    NO* aux = *fp;
    int tam = 0;
    while(aux != NULL){
        aux = aux->prox;
        tam++;
    }
    return tam;

}

void inserirPrioritario(FilaP* fp, int elem, int prio){

    if(fp == NULL) return;
    NO* n = alocarNO();
    n->info = elem;
    n->prio = prio;
    if(estaVazia(fp)){
        *fp = n;
        n->prox = NULL;
    }

    else{
        NO* aux = *fp;
        NO* ant = NULL;
        while(aux != NULL && aux->prio >= n->prio){
            ant = aux;
```

•



```c
}

void inserirPrioritario(FilaP* fp, int elem, int prio){

    if(fp == NULL) return;
    NO* n = alocarNO();
    n->info = elem;
    n->prio = prio;
    if(estaVazia(fp)){
        *fp = n;
        n->prox = NULL;
    }

    else{
        NO* aux = *fp;
        NO* ant = NULL;
        while(aux != NULL && aux->prio >= n->prio){
            ant = aux;
            aux = aux->prox;
        }
        if(ant == NULL){
            n->prox = *fp;
            *fp = n;
        }
        else{
            n->prox = ant->prox;
            ant->prox = n;
        }
    }
}

void removeInicio(FilaP* fp){
    if(fp == NULL) return;
    if(estaVazia(fp)) return;
    NO* aux = *fp;
    *fp = aux->prox;
    destroiNO(aux);
}

void verInicio(FilaP* fp, int* p){
    if(fp == NULL) return;
    if(estaVazia(fp)) return;
    *p = (*fp)->info;
}

void imprimeFilaP(FilaP* fp){
    if(fp == NULL){
        printf("Fila de Prioridade não encontrada.\n");
        return;
    }
    if(estaVazia(fp)){
        printf("Fila de prioridade vazia.\n");
        return;
    }
    NO* aux = *fp;
    printf("Elementos: ");
    while(aux != NULL){
        printf(" [%d, %d]", aux->info, aux->prio);
        aux = aux->prox;
    }
    printf("\n");
}

#endif
```

•


[lucascosta@fedora 2.1]$ ./main

Fila de Prioridade Simplesmente Encadeada:
1 - Criar Fila;
2 - Inserir um item pela prioridade;
3 - Ver o início da Fila;
4 - Remover um item;
5 - Imprimir a Fila;
6 - Mostrar o tamanho da Fila;
7 - Destruir a Fila;
8 - Sair.

1
2
Elemento a ser inserido: 85
Prioridade do elemento: 1
2
Elemento a ser inserido: 45
Prioridade do elemento: 3
5
Elementos: [45, 3] [85, 1]
6
Tamanho da Fila: 2
3
Início da Fila: 45
4
5
Elementos: [85, 1]
8
[lucascosta@fedora 2.1]$

.

2.2

2.2 > C main.c > main()

```c
#include <stdio.h>
#include <stdlib.h>
#include "fila.h"

int main(){

    FilaP* fp = NULL;
    int n, elem, pri;
    int* info = (int*)malloc(sizeof(int));
    int* prio = (int*)malloc(sizeof(int));

    printf("-----------------------------------------------\n");
    printf("Fila de Prioridade Simplesmente Encadeada:\n");
    printf("1 - Criar Fila;\n");
    printf("2 - Inserir um item pela prioridade;\n");
    printf("3 - Ver o início da Fila;\n");
    printf("4 - Remover um item;\n");
    printf("5 - Imprimir a Fila;\n");
    printf("6 - Mostrar o tamanho da Fila;\n");
    printf("7 - Destruir a Fila;\n");
    printf("8 - Sair.\n");
    printf("-----------------------------------------------\n");

    do{
        scanf("%d", &n);
        switch(n){
            case 1:
                fp = criaFilaP();
                break;
            case 2:
                printf("Elemento a ser inserido: ");
                scanf("%d", &elem);
                printf("Prioridade do elemento: ");
                scanf("%d", &pri);
                inserePrioritario(fp, elem, pri);
                break;
            case 3:
                verInicio(fp, info, prio);
                printf("Início da Fila de Prioridade: [%d, %d]\n", *info, *prio);
                break;
            case 4:
                removeInicio(fp);
                break;
            case 5:
                imprimeFilaP(fp);
                break;
            case 6:
                printf("Tamanho da Fila: %d\n", tamanhoFilaP(fp));
                break;
            case 7:
                destroiFilaP(fp);
                break;
        }
    } while(n != 8);
    free(info);
    free(prio);
    return 0;

```

2.2 > C fila.h > alocarNO()

```c
#ifndef FILAPRIO_H
#define FILAPRIO_H

#include <stdio.h>
#include <stdlib.h>
#define MAX 100

typedef struct{

    int info, prio;

} NO;

NO* alocarNO(){
    NO* n = (NO*)malloc(sizeof(NO));

    if(n != NULL){
        n->info = 0;
        n->prio = 0;
    }
    return n;

}

void destroiNO(NO* n){
    if(n == NULL) return;
    free(n);
}

typedef struct{

    int qtd;
    NO dados[MAX];

} FilaP;

FilaP* criaFilaP(){

    FilaP* fp = (FilaP*)malloc(sizeof(FilaP));
    if(fp != NULL){
        fp->qtd = 0;
    }
    return fp;

}

void destroiFilaP(FilaP* fp){

    if(fp == NULL) return;
    free(fp);


}

int tamanhoFilaP(FilaP* fp){

    if(fp == NULL) return -1;
    return fp->qtd;

```

.

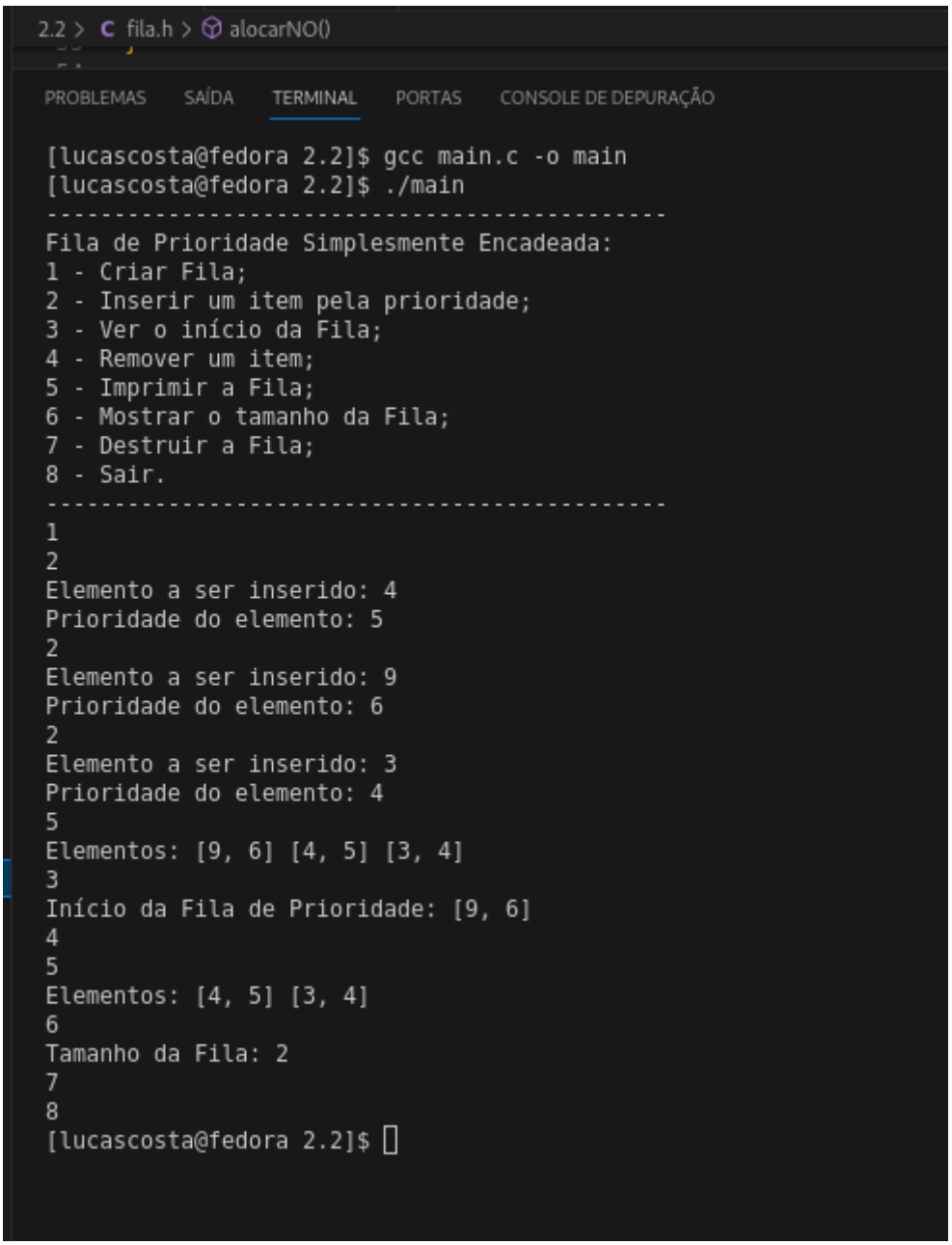
2.2 > C fila.h > alocarNO()

```c
int tamanhoFilaP(FilaP* fp){

    if(fp == NULL) return -1;
    return fp->qtd;

}

int estaVazia(FilaP* fp){
    if(fp == NULL) return -1;
    return (fp->qtd == 0);
}

int estaCheia(FilaP* fp){
    if(fp == NULL) return -1;
    return (fp->qtd == MAX);
}

void imprimeFilaP(FilaP* fp){

    if(fp == NULL){
        printf("Fila de Prioridade não encontrada.\n");
        return;
    }

    if(estaVazia(fp)){
        printf("Fila de Prioridade vazia.\n");
        return;
    }

    printf("Elementos:");
    for(int i = 0; i < fp->qtd; i++){
        printf(" [%d, %d]", fp->dados[i].info, fp->dados[i].prio);
    }
    printf("\n");

}

void trocarNO(NO* n, NO* m){

    NO* aux = alocarNO();
    if(aux == NULL) return;
    aux->info = n->info;
    aux->prio = n->prio;
    n->info = m->info;
    n->prio = m->prio;
    m->info = aux->info;
    m->prio = aux->prio;
    destroiNO(aux);

}

void ajustaHeapInsere(FilaP* fp, int filho){

    NO* aux = alocarNO();
    if(aux == NULL) return;
    int pai = (filho-1)/2;
    int prioPai = fp->dados[pai].prio;
```

2.2 > C fila.h > trocarNO(NO *, NO *)

```c
void ajustaHeapRemove(FilaP* fp, int pai){

    NO* aux = alocarNO();
    if(aux == NULL) return;

    int filho = (2*pai)+1;

    while(filho < fp->qtd){

        if(filho < fp->qtd-1){
            if(fp->dados[filho].prio < fp->dados[filho+1].prio){
                filho++;
            }
        }

        if(fp->dados[pai].prio > fp->dados[filho].prio){
            break;
        }

        trocarNO(&fp->dados[pai], &fp->dados[filho]);
        pai = filho;
        filho = (2*pai)+1;

    }
    destroiNO(aux);
}

void removeInicio(FilaP* fp){
    if(fp == NULL) return;
    if(estaVazia(fp)) return;

    fp->qtd--;
    fp->dados[0].info = fp->dados[fp->qtd].info;
    fp->dados[0].prio = fp->dados[fp->qtd].prio;
    ajustaHeapRemove(fp, 0);
}

#endif
```

.


2.2 > fila.h > alocarNO()
PROBLEMAS
SAÍDA
TERMINAL
PORTAS
CONSOLE DE DEPURAÇÃO
[lucascosta@fedora 2.2]$ gcc main.c -o main
[lucascosta@fedora 2.2]$ ./main

Fila de Prioridade Simplesmente Encadeada:
1 - Criar Fila;
2 - Inserir um item pela prioridade;
3 - Ver o início da Fila;
4 - Remover um item;
5 - Imprimir a Fila;
6 - Mostrar o tamanho da Fila;
7 - Destruir a Fila;
8 - Sair.

1
2
Elemento a ser inserido: 4
Prioridade do elemento: 5
2
Elemento a ser inserido: 9
Prioridade do elemento: 6
2
Elemento a ser inserido: 3
Prioridade do elemento: 4
5
Elementos: [9, 6] [4, 5] [3, 4]
3
Início da Fila de Prioridade: [9, 6]
4
5
Elementos: [4, 5] [3, 4]
6
Tamanho da Fila: 2
7
8
[lucascosta@fedora 2.2]$