

## Roteiro 6 - LUCAS EDUARDO LEITE COSTA

### 1.1

```
C main.c x C deque.c C deque.h 1.1 > C main.c > main()
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "deque.h"
4 #include "deque.h"
5
6 int main()
7
8     Deque* d;
9     int n, elem;
10    int* p = (int*)malloc(sizeof(int));
11
12    printf("-----\n");
13    printf("1 - Criar Deque\n");
14    printf("2 - Inserir um item no fim\n");
15    printf("3 - Inserir um item no inicio\n");
16    printf("4 - Ver o inicio do deque\n");
17    printf("5 - Ver o fim do deque\n");
18    printf("6 - Remover um item do fim\n");
19    printf("7 - Remover um item do inicio\n");
20    printf("8 - Imprimir o deque\n");
21    printf("9 - Destruir o deque\n");
22    printf("10 - Sair.\n");
23    printf("-----\n");
24
25    do{
26
27        scanf("%d", &n);
28
29        switch(n){
30
31            case 1:
32                d = criaDeque();
33                break;
34
35            case 2:
36                scanf("%d", &elem);
37                insereFim(d, elem);
38                break;
39
40            case 3:
41                scanf("%d", &elem);
42                insereInicio(d, elem);
43                break;
44
45            case 4:
46                verInicio(d, p);
47                printf("Inicio do deque: %d\n", *p);
48                break;
```

```
C main.c C deque.c x C deque.h 1.1 > C deque.c > ...
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "deque.h"
4 #define MAX 100
5
6 Deque* criaDeque()
7
8     Deque* d = (Deque*)malloc(sizeof(Deque));
9
10    if(d != NULL){
11        d->qtd = 0;
12        d->ini = 0;
13        d->fim = 0;
14    }
15    return d;
16
17 void destruiDeque(Deque* d){
18
19     if(d == NULL) return;
20     free(d);
21 }
22
23 int tamanhoDeque(Deque* d){
24
25     if(d == NULL) return -1;
26     return d->qtd;
27 }
28
29 int estaCheio(Deque* d){
30
31     if(d == NULL) return -1;
32     return (d->qtd == MAX);
33 }
34
35 int estaVazio(Deque* d){
36
37     if(d == NULL) return -1;
38     return (d->qtd == 0);
39 }
40
41 void insereFim(Deque* d, int elem){
42
43     if(d == NULL) return;
44     if(estaCheio(d)) return;
```

```
C main.c x deque.c x deque.h ...
1.1 > C deque.c > CriaDeque()
85 }
86
87 void verInicio(Deque* d, int* p){
88
89     if(d == NULL) return;
90     if(estaVazio(d)) return;
91
92     *p = d->dados[d->ini];
93
94 }
95
96 void verFim(Deque* d, int* p){
97
98     if(d == NULL) return;
99     if(estaVazio(d)) return;
100
101     int i = (d->fim-1 < 0 ? MAX-1 : d->fim-1);
102     *p = d->dados[i];
103 }
104
105 void imprimeDeque(Deque* d){
106
107     if(d == NULL){
108         printf("Deque não encontrado.\n");
109         return;
110     }
111
112     if(estaVazio(d)){
113         printf("Deque vazio.\n");
114         return;
115     }
116
117     int i = d->ini;
118     printf("Elementos: ");
119     do{
120         printf(" %d", d->dados[i]);
121         i = (i+1) % MAX;
122     } while(i != d->fim);
123     printf("\n");
124 }
125
126
1.1 > C deque.h > C destróiDeque(Deque*)
1 #ifndef DEQUE_H
2 #define DEQUE_H
3 #include <stdio.h>
4 #include <stdlib.h>
5 #define MAX 100
6
7 typedef struct{
8
9     int qtd, ini, fim;
10     int dados[MAX];
11 }Deque;
12
13
14 Deque* criaDeque();
15 void destróiDeque(Deque* d);
16 int tamanhoDeque(Deque* d);
17 int estaCheio(Deque* d);
18 int estaVazio(Deque* d);
19 void insereFim(Deque* d, int elem);
20 void insereInicio(Deque* d, int elem);
21 void removeFim(Deque* d);
22 void removeInicio(Deque* d);
23 void verInicio(Deque* d, int* p);
24 void verFim(Deque* d, int* p);
25 void imprimeDeque(Deque* d);
26
27 #endif
Ln 15, Col 5 - Espacos: 4 - UTF-8 - LF - {}
```

```
[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

```

12 > C main.c > ...
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "deque.h"
4
5  int main(){
6
7      Deque* d;
8      int n, elem;
9      int* p = (int*)malloc(sizeof(int));
10
11      printf("-----\n");
12      printf("1 - Criar Deque\n");
13      printf("2 - Inserir um item no fim\n");
14      printf("3 - Inserir um item no inicio\n");
15      printf("4 - Ver o inicio do deque\n");
16      printf("5 - Ver o fim do deque\n");
17      printf("6 - Remover um item do fim\n");
18      printf("7 - Remover um item do inicio\n");
19      printf("8 - Imprimir o deque\n");
20      printf("9 - Destruir o deque\n");
21      printf("10 - Sair.\n");
22      printf("-----\n");
23
24      do{
25
26          scanf("%d", &n);
27
28          switch(n){
29
30              case 1:
31                  d = criaDeque();
32                  break;
33
34              case 2:
35                  scanf("%d", &elem);
36                  insereFim(d, elem);
37                  break;
38
39              case 3:
40                  scanf("%d", &elem);
41                  insereInicio(d, elem);
42                  break;
43
44              case 4:
45                  verInicio(d, p);
46                  printf("Inicio do deque: %d\n", *p);
47                  break;
48
49              case 5:
50                  verFim(d, p);
51                  printf("Fim do deque: %d\n", *p);
52                  break;
53
54              case 6:
55                  removeFim(d);
56                  break;
57
58              case 7:
59                  removeInicio(d);
60                  break;
61
62              case 8:
63                  imprimeDeque(d);
64                  break;
65
66              case 9:
67                  destroiDeque(d);
68                  break;
69
70          }
71
72          } while(n != 10);
73      free(p);
74      return 0;
75  }
76

```

```

12 > C deque.c > ...
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "deque.h"
4
5  NO* alocarNO(){
6      NO* n = (NO*)malloc(sizeof(NO));
7      if(n != NULL){
8          n->info = 0;
9          n->prox = NULL;
10         n->ant = NULL;
11     }
12     return n;
13 }
14
15 void destroiNO(NO* n){
16     if(n == NULL) return;
17     free(n);
18 }
19
20
21 Deque* criaDeque(){
22     Deque* d = (Deque*)malloc(sizeof(Deque));
23     if(d != NULL){
24         d->qtd = 0;
25         d->ini = NULL;
26         d->fim = NULL;
27     }
28     return d;
29 }
30
31
32
33 void destroiDeque(Deque* d){
34     if(d == NULL) return;
35
36     NO* aux;
37     while(d->ini != NULL){
38         aux = d->ini;
39         d->ini = d->ini->prox;
40         free(aux);
41     }
42     free(d);
43 }
44
45
46 int tamanhoDeque(Deque* d){
47     if(d == NULL) return -1;
48

```

```

12 > C deque.c > alocarNO()
43 }
44
45
46 int tamanhoDeque(Deque* d){
47     if(d == NULL) return -1;
48     return d->qtd;
49 }
50
51
52
53 int estaVazio(Deque* d){
54     if(d == NULL) return -1;
55     return (d->qtd == 0);
56 }
57
58
59 void insereFim(Deque* d, int elem){
60     if(d == NULL) return;
61
62     NO* n = alocarNO();
63     if(n == NULL) return;
64
65     n->info = elem;
66     n->prox = NULL;
67
68     if(estaVazio(d)){
69         n->ant = NULL;
70         d->ini = n;
71     }
72
73     else{
74         d->fim->prox = n;
75         n->ant = d->fim;
76     }
77
78     d->fim = n;
79     d->qtd++;
80
81 }
82
83
84 void insereInicio(Deque* d, int elem){
85     if(d == NULL) return;
86
87     NO* n = alocarNO();
88     if(n == NULL) return;
89

```

```

1.2 > C deque.c > ...
83
84 void insereInicio(Deque* d, int elem){
85
86     if(d == NULL) return;
87
88     NO* n = alocarNO();
89     if(n == NULL) return;
90
91     n->info = elem;
92     n->ant = NULL;
93
94     if(estaVazio(d)){
95         n->prox = NULL;
96         d->fim = n;
97     }
98
99     else{
100         d->ini->ant = n;
101         n->prox = d->ini;
102     }
103
104     d->ini = n;
105     d->qtd++;
106 }
107
108 void removeFim(Deque* d){
109
110     if(d == NULL) return;
111     if(estaVazio(d)) return;
112     NO* aux = d->fim;
113     if(d->ini == d->fim){
114         d->ini = NULL;
115         d->fim = NULL;
116     }
117     else{
118         d->fim = d->fim->ant;
119         d->fim->prox = NULL;
120     }
121     destroiNO(aux);
122     d->qtd--;
123 }
124
125 void removeInicio(Deque* d){
126
127     if(d == NULL) return;
128     if(estaVazio(d)) return;
129     ...

```

```

1.2 > C deque.c > alocarNO()
125
126 void removeInicio(Deque* d){
127
128     if(d == NULL) return;
129     if(estaVazio(d)) return;
130     NO* aux = d->ini;
131     if(d->ini == d->fim){
132         d->ini = NULL;
133         d->fim = NULL;
134     }
135     else{
136         d->ini = d->ini->prox;
137         d->ini->ant = NULL;
138     }
139     destroiNO(aux);
140     d->qtd--;
141 }
142
143 void verInicio(Deque* d, int* p){
144
145     if(d == NULL) return;
146     if(estaVazio(d)) return;
147     *p = d->ini->info;
148 }
149
150 void verFim(Deque* d, int* p){
151
152     if(d == NULL) return;
153     if(estaVazio(d)) return;
154     *p = d->fim->info;
155 }
156
157 void imprimeDeque(Deque* d){
158
159     if(d == NULL){
160         printf("Deque não encontrado.\n");
161         return;
162     }
163
164     if(estaVazio(d)){
165         printf("Deque vazio.\n");
166         return;
167     }
168
169     NO* aux = d->ini;
170     printf("Elementos: ");

```

```

1.2 > C deque.c > ...
154     *p = d->fim->info;
155 }
156
157 void imprimeDeque(Deque* d){
158
159     if(d == NULL){
160         printf("Deque não encontrado.\n");
161         return;
162     }
163
164     if(estaVazio(d)){
165         printf("Deque vazio.\n");
166         return;
167     }
168
169     NO* aux = d->ini;
170
171     printf("Elementos:");
172     while(aux != d->fim->prox){
173         printf(" %d", aux->info);
174         aux = aux->prox;
175     }
176     printf("\n");
177 }
178
179 }
180

```

```

1.2 > C deque.h > __unnamed_struct_1f19_1 > fim
1
2 #ifndef DEQUE_H
3 #define DEQUE_H
4 #include <stdio.h>
5 #include <stdlib.h>
6
7 typedef struct NO NO;
8 struct NO{
9     int info;
10    NO* prox;
11    NO* ant;
12 };
13
14 typedef struct{
15     int qtd;
16     struct NO* ini;
17     struct NO* fim;
18 }Deque;
19
20 NO* alocarNO();
21 void destroiNO(NO* n);
22 Deque* criaDeque();
23 void destroiDeque(Deque* d);
24 int tamanhoDeque(Deque* d);
25 int estaVazio(Deque* d);
26 void insereFim(Deque* d, int elem);
27 void insereInicio(Deque* d, int elem);
28 void removeFim(Deque* d);
29 void removeInicio(Deque* d);
30 void verInicio(Deque* d, int* p);
31 void verFim(Deque* d, int* p);
32 void imprimeDeque(Deque* d);
33
34 #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()
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "fila.h"
4
5  int main(){
6
7      FilaP* fp = NULL;
8      int n, elem, prio;
9      int* p = (int*)malloc(sizeof(int));
10
11      printf("-----\n");
12      printf("Fila de Prioridade Simplesmente Encadeada:\n");
13      printf("1 - Criar Fila;\n");
14      printf("2 - Inserir um item pela prioridade;\n");
15      printf("3 - Ver o início da Fila;\n");
16      printf("4 - Remover um item;\n");
17      printf("5 - Imprimir a Fila;\n");
18      printf("6 - Mostrar o tamanho da Fila;\n");
19      printf("7 - Destruir a Fila;\n");
20      printf("8 - Sair.\n");
21      printf("-----\n");
22
23      do{
24          scanf("%d", &n);
25          switch(n){
26              case 1:
27                  fp = criaFilaP();
28                  break;
29              case 2:
30                  printf("Elemento a ser inserido: ");
31                  scanf("%d", &elem);
32                  printf("Prioridade do elemento: ");
33                  scanf("%d", &prio);
34                  inserirPrioritario(fp, elem, prio);
35                  break;
36              case 3:
37                  verInicio(fp, p);
38                  printf("Início da Fila: %d\n", *p);
39                  break;
40              case 4:
41                  removeInicio(fp);
42                  break;
43              case 5:
44                  imprimeFilaP(fp);
45                  break;
46              case 6:
47                  printf("Tamanho da Fila: %d\n", tamanhoFilaP(fp));
48                  break;
49              case 7:
50                  destroiFilaP(fp);
51                  break;
52          }
53      } while(n != 8);
54      free(p);
55      return 0;
56
57  }

```



```

2.1 > C filah > ...
1  #ifndef FILA_H
2  #define FILA_H
3
4  #include <stdio.h>
5  #include <stdlib.h>
6
7  typedef struct NO NO;
8
9  struct NO{
10     int info, prio;
11     NO* prox;
12 }
13
14 NO* alocarNO(){
15     NO* n = (NO*)malloc(sizeof(NO));
16     if(n != NULL){
17         n->info = 0;
18         n->prio = 0;
19         n->prox = NULL;
20     }
21     return n;
22 }
23
24 void destroiNO(NO* n){
25     if(n == NULL) return;
26     free(n);
27 }
28
29 typedef NO* Filap;
30
31 Filap* criaFilaP(){
32     Filap* fp = (Filap*)malloc(sizeof(Filap));
33     if(fp != NULL){
34         *fp = NULL;
35     }
36     return fp;
37 }
38
39 void destroiFilaP(Filap* fp){
40     if(fp == NULL) return;
41
42     NO* aux;
43     while((*fp) != NULL){
44         aux = *fp;
45         *fp = (*fp)->prox;
46         destroiNO(aux);
47     }
48 }

```

```

2.1 > C filah > alocarNO()
40
41 void destroiFilaP(Filap* fp){
42     if(fp == NULL) return;
43
44     NO* aux;
45     while((*fp) != NULL){
46         aux = *fp;
47         *fp = (*fp)->prox;
48         destroiNO(aux);
49     }
50     free(fp);
51 }
52
53
54 int estaVazia(Filap* fp){
55     if(fp == NULL) return -1;
56     return (*fp == NULL);
57 }
58
59 int tamanhoFilaP(Filap* fp){
60     if(fp == NULL) return -1;
61     if(estaVazia(fp)) return 0;
62     NO* aux = *fp;
63     int tam = 0;
64     while(aux != NULL){
65         aux = aux->prox;
66         tam++;
67     }
68     return tam;
69 }
70
71 void inserirPrioritario(Filap* fp, int elem, int prio){
72     if(fp == NULL) return;
73     NO* n = alocarNO();
74     n->info = elem;
75     n->prio = prio;
76     if(estaVazia(fp)){
77         *fp = n;
78         n->prox = NULL;
79     }
80     else{
81         NO* aux = *fp;
82         NO* ant = NULL;
83         while(aux != NULL && aux->prio >= n->prio){
84             ant = aux;
85             aux = aux->prox;
86         }
87     }

```

```

2.1 > C filah > ...
69
70 }
71
72 void inserirPrioritario(Filap* fp, int elem, int prio){
73
74     if(fp == NULL) return;
75     NO* n = alocarNO();
76     n->info = elem;
77     n->prio = prio;
78     if(estaVazia(fp)){
79         *fp = n;
80         n->prox = NULL;
81     }
82     else{
83         NO* aux = *fp;
84         NO* ant = NULL;
85         while(aux != NULL && aux->prio >= n->prio){
86             ant = aux;
87             aux = aux->prox;
88         }
89         if(ant == NULL){
90             n->prox = *fp;
91             *fp = n;
92         }
93         else{
94             n->prox = ant->prox;
95             ant->prox = n;
96         }
97     }
98 }
99
100 void removeInicio(Filap* fp){
101     if(fp == NULL) return;
102     if(estaVazia(fp)) return;
103     NO* aux = *fp;
104     *fp = aux->prox;
105     destroiNO(aux);
106 }
107
108 void verInicio(Filap* fp, int* p){
109     if(fp == NULL) return;
110     if(estaVazia(fp)) return;
111     *p = (*fp)->info;
112 }
113
114 void imprimeFilaP(Filap* fp){
115     if(fp == NULL){
116

```

```

2.1 > C filah > alocarNO()
98
99 }
100
101 void removeInicio(Filap* fp){
102     if(fp == NULL) return;
103     if(estaVazia(fp)) return;
104     NO* aux = *fp;
105     *fp = aux->prox;
106     destroiNO(aux);
107 }
108
109 void verInicio(Filap* fp, int* p){
110     if(fp == NULL) return;
111     if(estaVazia(fp)) return;
112     *p = (*fp)->info;
113 }
114
115 void imprimeFilaP(Filap* fp){
116     if(fp == NULL){
117         printf("Fila de Prioridade não encontrada.\n");
118         return;
119     }
120     if(estaVazia(fp)){
121         printf("Fila de prioridade vazia.\n");
122         return;
123     }
124     NO* aux = *fp;
125     printf("Elementos: ");
126     while(aux != NULL){
127         printf(" [%d, %d]", aux->info, aux->prio);
128         aux = aux->prox;
129     }
130     printf("\n");
131 }
132
133 #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

```

22 > C main.c > @ main()
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "fila.h"
4
5 int main(){
6
7     FilaP* fp = NULL;
8     int n, elem, pri;
9     int* info = (int*)malloc(sizeof(int));
10    int* prio = (int*)malloc(sizeof(int));
11
12    printf("-----\n");
13    printf("Fila de Prioridade Simplesmente Encadeada:\n");
14    printf("1 - Criar Fila;\n");
15    printf("2 - Inserir um item pela prioridade;\n");
16    printf("3 - Ver o início da Fila;\n");
17    printf("4 - Remover um item;\n");
18    printf("5 - Imprimir a Fila;\n");
19    printf("6 - Mostrar o tamanho da Fila;\n");
20    printf("7 - Destruir a Fila;\n");
21    printf("8 - Sair.\n");
22    printf("-----\n");
23
24    do{
25        scanf("%d", &n);
26        switch(n){
27            case 1:
28                fp = criaFilaP();
29                break;
30            case 2:
31                printf("Elemento a ser inserido: ");
32                scanf("%d", &elem);
33                printf("Prioridade do elemento: ");
34                scanf("%d", &pri);
35                inserePrioritario(fp, elem, pri);
36                break;
37            case 3:
38                verInicio(fp, info, prio);
39                printf("Início da Fila de Prioridade: [%d, %d]\n", *info, *prio);
40                break;
41            case 4:
42                removeInicio(fp);
43                break;
44            case 5:
45                imprimeFilaP(fp);
46                break;
47            case 6:
48                printf("Tamanho da Fila: %d\n", tamanhoFilaP(fp));
49                break;
50            case 7:
51                destroiFilaP(fp);
52                break;
53        }
54    } while(n != 8);
55    free(info);
56    free(prio);
57    return 0;
58
22 > C fila.h > @ alocarNO()
1 #ifndef FILAPRIO_H
2 #define FILAPRIO_H
3
4 #include <stdio.h>
5 #include <stdlib.h>
6 #define MAX 100
7
8 typedef struct{
9     int info, prio;
10 } NO;
11
12 NO* alocarNO(){
13     NO* n = (NO*)malloc(sizeof(NO));
14
15     if(n != NULL){
16         n->info = 0;
17         n->prio = 0;
18     }
19     return n;
20 }
21
22 void destroiNO(NO* n){
23     if(n == NULL) return;
24     free(n);
25 }
26
27 typedef struct{
28     int qtd;
29     NO dados[MAX];
30 } FilaP;
31
32 FilaP* criaFilaP(){
33     FilaP* fp = (FilaP*)malloc(sizeof(FilaP));
34     if(fp != NULL){
35         fp->qtd = 0;
36     }
37     return fp;
38 }
39
40 void destroiFilaP(FilaP* fp){
41     if(fp == NULL) return;
42     free(fp);
43 }
44
45 int tamanhoFilaP(FilaP* fp){
46     if(fp == NULL) return -1;
47     return fp->qtd;
48 }

```

```

22 > C fila.h > @ alocarNO()
54
55 int tamanhoFilaP(FilaP* fp){
56     if(fp == NULL) return -1;
57     return fp->qtd;
58 }
59
60
61
62 int estaVazia(FilaP* fp){
63     if(fp == NULL) return -1;
64     return (fp->qtd == 0);
65 }
66
67
68 int estaCheia(FilaP* fp){
69     if(fp == NULL) return -1;
70     return (fp->qtd == MAX);
71 }
72
73 void imprimeFilaP(FilaP* fp){
74     if(fp == NULL){
75         printf("Fila de Prioridade não encontrada.\n");
76         return;
77     }
78
79     if(estaVazia(fp)){
80         printf("Fila de Prioridade vazia.\n");
81         return;
82     }
83
84     printf("Elementos:");
85     for(int i = 0; i < fp->qtd; i++){
86         printf(" [%d, %d]", fp->dados[i].info, fp->dados[i].prio);
87     }
88     printf("\n");
89 }
90
91
92 void trocarNO(NO* n, NO* m){
93
94     NO* aux = alocarNO();
95     if(aux == NULL) return;
96     aux->info = n->info;
97     aux->prio = n->prio;
98     n->info = m->info;
99     n->prio = m->prio;
100    m->info = aux->info;
101    m->prio = aux->prio;
102    destroiNO(aux);
103 }
104
105
106 void ajustaHeapInserir(FilaP* fp, int filho){
107
108     NO* aux = alocarNO();
109     if(aux == NULL) return;
110     int pai = (filho-1)/2;
111     int prioPai = fp->dados[pai].prio;

```

```

22 > C fila.h > @ trocarNO(NO* n, NO* m)
143
144 void ajustaHeapRemove(FilaP* fp, int pai){
145
146     NO* aux = alocarNO();
147     if(aux == NULL) return;
148
149     int filho = (2*pai)+1;
150
151     while(filho < fp->qtd){
152
153         if(filho < fp->qtd-1){
154             if(fp->dados[filho].prio < fp->dados[filho+1].prio){
155                 filho++;
156             }
157         }
158
159         if(fp->dados[pai].prio > fp->dados[filho].prio){
160             break;
161         }
162
163         trocarNO(&fp->dados[pai], &fp->dados[filho]);
164         pai = filho;
165         filho = (2*pai)+1;
166     }
167
168     destroiNO(aux);
169 }
170
171 void removeInicio(FilaP* fp){
172     if(fp == NULL) return;
173     if(estaVazia(fp)) return;
174
175     fp->qtd--;
176     fp->dados[0].info = fp->dados[fp->qtd].info;
177     fp->dados[0].prio = fp->dados[fp->qtd].prio;
178     ajustaHeapRemove(fp, 0);
179 }
180
181 #endif

```

2.2 > C 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]$
```