deque_sequencial_estatico.h

```
1 #ifndef DSE H
 2
    #define DSE H
 3
    #include <stdio.h>
    #include <stdlib.h>
 5
    #define MAX 100
 6
 7
    typedef struct{
 8
        int dados[MAX];
 9
        int qtd, inicio, fim;
10
    }Deque;
11
12
    Deque* criaDeque(){
        Deque* deque;
13
        deque = (Deque*)malloc(sizeof(Deque));
14
15
        if (deque != NULL){
16
            deque->fim = 0;
17
            deque->inicio = 0;
18
            deque->qtd = 0;
19
20
        return deque;
21
    }
22
    void destroiDeque(Deque* deque){
23
        if (deque != NULL)
24
            free(deque);
25
26
27
28
    int tamanhoDeque(Deque* deque){
29
        if (deque == NULL)
30
            return -1;
31
        return deque->qtd;
32
33
34
    int estaCheio(Deque* deque){
35
        if (deque == NULL)
36
            return -1;
37
        return (deque->qtd == MAX);
38
39
40
    int estaVazio(Deque* deque){
        if (deque == NULL)
41
42
            return -1;
43
        return (deque->qtd == 0);
44
    }
45
    int insereFim(Deque* deque, int elemento){
46
47
        if (deque == NULL)
            return 0;
48
49
        if (estaCheio(deque))
50
            return 0;
51
        deque->dados[deque->fim] = elemento;
52
        deque->fim = (deque->fim+1) % MAX;
53
        deque->qtd++;
54
        return 1;
55
    }
56
    int insereInicio(Deque* deque, int elemento){
```

```
58
         if (deque == NULL)
 59
             return 0;
         if (estaCheio(deque))
 60
 61
             return 0;
 62
         deque->inicio = (deque->inicio - 1 < 0 ? MAX -1 : deque->inicio-1);
 63
         deque->dados[deque->inicio] = elemento;
 64
         deque->qtd++;
 65
         return 1;
 66
 67
     int removeFim(Deque* deque){
 68
 69
         if (deque == NULL)
 70
             return 0;
 71
         if (estaVazio(deque))
 72
             return 0;
 73
         deque->fim = (deque->fim-1 < 0 ? MAX-1 : deque->fim-1);
 74
         deque->qtd--;
 75
         return 1;
 76
     }
 77
     int removeInicio(Deque* deque){
 78
 79
         if (deque == NULL)
 80
             return 0;
 81
         if (estaVazio(deque))
 82
             return 0;
 83
         deque->inicio = (deque->inicio+1)%MAX;
 84
         deque->qtd--;
 85
         return 1;
 86
 87
     int verInicio (Deque* deque, int* elemento){
 88
 89
         if (deque == NULL)
 90
             return 0;
 91
         if (estaVazio(deque))
 92
             return 0;
93
         *elemento = deque->dados[deque->inicio];
 94
         return 1;
 95
     }
96
     int verFim(Deque* deque, int* elemento){
 97
 98
         if (deque == NULL)
99
             return 0;
         if (estaVazio(deque))
100
101
             return 0;
102
         int i = (deque->fim-1 < 0 ? MAX-1 : deque->fim-1);
         *elemento = deque->dados[i];
103
104
         return 1;
105
106
107
     void imprime(Deque* deque){
108
         if (deque == NULL)
109
             return;
110
         if (estaVazio(deque)){
111
             printf ("Deque vazio!\n");
112
             return;
113
         }
114
         int i = deque->inicio;
         printf ("Elementos: \n");
115
116
         do{
117
             printf ("%d ", deque->dados[i]);
```

deque_duplamente_encadeado.h

```
#ifndef DDE H
 2
   #define DDE H
 3
   #include <stdio.h>
 4
    #include <stdlib.h>
 5
   typedef struct NO{
 6
 7
        int info;
 8
        struct NO *proximo;
 9
        struct NO *anterior;
10
    } NO;
    typedef struct{
11
12
        int qtd;
        struct NO *inicio;
13
        struct NO *fim;
14
15
    } Deque;
16
17
    NO *alocarNO()
18
    {
19
        return (NO *)malloc(sizeof(NO));
20
21
22
    void liberarNO(NO *q){
23
        free(q);
24
25
26
    Deque *criaDeque(){
        Deque *deque;
27
28
        deque = (Deque *)malloc(sizeof(Deque));
29
        if (deque != NULL){
30
            deque \rightarrow qtd = 0;
            deque->inicio = NULL;
32
            deque->fim = NULL;
33
34
        return deque;
35
    }
36
    void destroiDeque(Deque *deque){
37
        if (deque != NULL){
38
39
            NO *auxiliar;
40
            while (deque->inicio != NULL)
41
42
                 auxiliar = deque->inicio;
43
                 deque->inicio = deque->inicio->proximo;
44
                 liberarNO(auxiliar);
45
46
            free(deque);
47
        }
48
49
50
    int tamanhoDeque(Deque *deque){
51
        if (deque == NULL)
52
            return -1;
53
        return deque->qtd;
    }
54
55
56
    int estaVazio(Deque *deque){
57
        if (deque == NULL)
```

```
58
             return -1;
 59
         return (deque->qtd == 0);
 60
 61
 62
     int insereInicio(Deque *deque, int elem){
 63
         if (deque == NULL)
 64
             return 0;
 65
         NO *novo = alocarNO();
         if (novo == NULL)
 66
 67
             return 0;
 68
         novo->info = elem;
         novo->anterior = NULL;
 69
 70
         if (estaVazio(deque)){
 71
             novo->proximo = NULL;
 72
             deque->fim = novo;
 73
 74
         else{
 75
             deque->inicio->anterior = novo;
 76
             novo->proximo = deque->inicio;
 77
 78
         deque->inicio = novo;
 79
         deque->qtd++;
 80
         return 1;
 81
     }
 82
 83
     int insereFim(Deque *deque, int elem){
 84
         if (deque == NULL)
 85
             return 0;
 86
         NO *novo = alocarNO();
 87
         if (novo == NULL)
 88
             return 0;
 89
         novo->info = elem;
 90
         novo->proximo = NULL;
 91
         if (estaVazio(deque)){
 92
              novo->anterior = NULL;
 93
             deque->inicio = novo;
 94
         }
 95
         else{
 96
             deque->fim->proximo = novo;
             novo->anterior = deque->fim;
 97
 98
99
         deque->fim = novo;
         deque->qtd++;
100
101
         return 1;
102
     }
103
104
     int removeInicio(Deque *deque){
         if (deque == NULL)
105
106
             return 0;
107
         if (estaVazio(deque))
108
             return 0;
         NO *auxiliar = deque->inicio;
109
         if (deque->inicio == deque->fim){
110
111
              deque->inicio = deque->fim = NULL;
112
113
         else{
              deque->inicio = deque->inicio->proximo;
114
115
              deque->inicio->anterior = NULL;
116
117
         liberarNO(auxiliar);
```

```
118
         deque->qtd--;
119
         return 1;
120
121
122
     int removeFim(Deque *deque){
         if (deque == NULL)
123
124
             return 0;
125
         if (estaVazio(deque))
126
             return 0;
127
         NO *auxiliar = deque->fim;
128
         if (deque->inicio == deque->fim){
129
             deque->inicio = deque->fim = NULL;
130
131
         else{
132
             deque->fim = deque->fim->anterior;
133
             deque->inicio->proximo = NULL;
134
135
         liberarNO(auxiliar);
136
         deque->qtd--;
137
         return 1;
138
     }
139
     int verInicio(Deque *deque, int *p){
140
141
         if (deque == NULL)
             return 0;
142
143
         if (estaVazio(deque))
144
             return 0;
145
         *p = deque->inicio->info;
146
         return 1;
147
     }
148
149
     int verFim(Deque *deque, int *p){
150
         if (deque == NULL)
151
             return 0;
         if (estaVazio(deque))
152
153
             return 0;
154
         *p = deque->fim->info;
155
         return 1;
156
     }
157
     void imprime(Deque *deque){
158
         if (deque == NULL)
159
160
             return;
161
         if (estaVazio(deque)){
162
             printf("Deque Vazio!\n");
163
             return;
164
         NO *auxiliar = deque->inicio;
165
166
         printf("Elementos:\n");
         while (auxiliar != NULL){
167
             printf("%d ", auxiliar->info);
168
             auxiliar = auxiliar->proximo;
169
170
         printf("\n");
171
172
173
     #endif // DDE H
174
```

13/10/2023, 14:02 exercicio1.c

exercicio1.c

```
1 #include <stdio.h>
   #include "deque_sequencial_estatico.h"
   // #include "deque_duplamente_encadeado.h"
 3
4
 5
   Esse arquivo funciona tanto para o Deque Sequencial Estático, quanto para o Deque
   Duplamente Encadeado.
    É necessário apenas importar a biblioteca que for utilizar
 6
7
        - deque sequencial estatico.h
8
9
        deque_duplamente_encadeado.h
    */
10
11
12
    int main(){
13
        int option, elemento;
14
        Deque* deque = NULL;
15
        do{
16
17
            printf ("1- Criar Deque.\n");
            printf ("2- Inserir item no inicio.\n");
18
19
            printf ("3- Inserir item no fim.\n");
            printf ("4- Ver o inicio do deque.\n");
20
21
            printf ("5- Ver o fim do deque.\n");
22
            printf ("6- Remover item do fim.\n");
            printf ("7- Remover item do inicio.\n");
23
24
            printf ("8- Imprimir deque.\n");
25
            printf ("9- Destruir deque.\n");
26
            printf ("10- Sair.\n");
27
            printf ("Opcao: ");
28
29
            scanf ("%d", &option);
30
            switch (option){
31
32
                case 1:
33
                    if (deque != NULL){
34
                         destroiDeque(deque);
                        printf ("Deque resetado!\n");
35
36
                    deque = criaDeque();
37
                    break;
38
39
                case 2:
                    printf ("Digite o numero que deseja adicionar no inicio do deque: \n");
40
41
                    scanf ("%d", &elemento);
                    if (insereInicio(deque, elemento)){
42
43
                        printf ("Adicionou (%d) no inicio!\n", elemento);
44
45
                        printf ("Nao foi possivel adicionar!\n");
46
47
                    break;
                case 3:
48
49
                    printf ("Digite o numero que deseja adicionar no fim do deque: \n");
                    scanf ("%d", &elemento);
50
                    if (insereFim(deque, elemento)){
51
52
                         printf ("Adicionou (%d) no fim!\n", elemento);
53
                    } else{
54
                        printf ("Nao foi possivel adicionar!\n");
55
56
                    break;
```

```
57
                case 4:
                     if (verInicio(deque, &elemento)){
58
59
                         printf ("Inicio do Deque = %d\n", elemento);
60
                     } else{
                         printf ("Nao foi possivel ver o inicio.\n");
61
62
                    break;
63
                case 5:
64
65
                     if (verFim(deque, &elemento)){
                         printf ("Fim do Deque = %d\n", elemento);
66
67
                     } else{
                         printf ("Nao foi possivel ver o Fim.\n");
68
69
                     }
70
                    break;
71
                case 6:
                     removeFim(deque);
72
73
                    break;
74
                case 7:
75
                     removeInicio(deque);
76
                    break;
77
                case 8:
                    imprime(deque);
78
79
                    break;
80
                case 9:
                    if (deque != NULL)
81
82
                         destroiDeque(deque);
83
                    break;
84
                case 10:
85
                    if (deque != NULL)
86
                         destroiDeque(deque);
                    printf ("Saindo do menu!\n");
87
88
                     break;
89
                default:
90
                    printf ("Opcao invalida! Tente de novo.\n");
91
        } while (option != 10);
92
93
        return 0;
94 }
```

1- Criar Deque. 2- Inserir item no inicio. 3- Inserir item no fim. 4- Ver o inicio do deque. 5- Ver o fim do deque.	 Criar Deque. Inserir item no inicio. Inserir item no fim. Ver o inicio do deque. Ver o fim do deque. 	1- Criar Deque.2- Inserir item no inicio.3- Inserir item no fim.4- Ver o inicio do deque.5- Ver o fim do deque.
6- Remover item do fim.	6- Remover item do fim.	6- Remover item do fim.
7- Remover item do inicio.	7- Remover item do inicio.	7- Remover item do inicio.
8- Imprimir deque.	8- Imprimir deque.	8- Imprimir deque.
9- Destruir deque.	9- Destruir deque.	9- Destruir deque.
10- Sair.	10- Sair.	10- Sair.
Opcao: 1	Opcao: 4	Opcao: 8
1- Criar Deque.	Inicio do Deque = 10	Elementos:
2- Inserir item no inicio.	1- Criar Deque.	20
3- Inserir item no fim.	2- Inserir item no inicio.	1- Criar Deque.
4- Ver o inicio do deque.	3- Inserir item no fim.	2- Inserir item no inicio.
5- Ver o fim do deque.	4- Ver o inicio do deque.	3- Inserir item no fim.
6- Remover item do fim.	5- Ver o fim do deque.	4- Ver o inicio do deque.
7- Remover item do inicio.	6- Remover item do fim.	5- Ver o fim do deque.
8- Imprimir deque.	7- Remover item do inicio.	6- Remover item do fim.
9- Destruir deque.	8- Imprimir deque.	7- Remover item do inicio.
10- Sair.	9- Destruir deque.	8- Imprimir deque.
Opcao: 2	10- Sair.	9- Destruir deque.
Digite o numero que deseja adicionar no inicio do deque: -	Opcao: 5 Fim do Deque = 20	10- Sair. Opcao: 9
Adicionou (10) no inicio!	1- Criar Deque.	1- Criar Deque.
1- Criar Deque.	2- Inserir item no inicio.	2- Inserir item no inicio.
2- Inserir item no inicio.	3- Inserir item no fim.	3- Inserir item no fim.
3- Inserir item no fim.	4- Ver o inicio do deque.	4- Ver o inicio do deque.
4- Ver o inicio do deque.	5- Ver o fim do deque.	5- Ver o fim do deque.
5- Ver o fim do deque.	6- Remover item do fim.	6- Remover item do fim.
6- Remover item do fim.	7- Remover item do inicio.	7- Remover item do inicio.
7- Remover item do inicio.	8- Imprimir deque.	8- Imprimir deque.
8- Imprimir deque.	9- Destruir deque.	9- Destruir deque.
9- Destruir deque.	10- Sair.	10- Sair.
10- Sair. Opcao: 3	Opcao: 7	Opcao: 10
Digite o numero que deseja adicionar no fim do deque:		Saindo do menu!
20		
Adicionou (20) no fim!		
(20) 110 12111		

fila_prioridades_simplesmente_encadeada.h

```
#ifndef FPSE H
 2
    #define FPSE H
 3
    #include <stdio.h>
 4
    #include <stdlib.h>
 5
 6
    typedef struct NO{
 7
        int info, prioridade;
 8
        struct NO* proximo;
 9
    typedef struct NO* FilaPrioridade;
10
11
12
    FilaPrioridade* criaFila(){
13
        FilaPrioridade* filaP;
        filaP = (FilaPrioridade*)malloc(sizeof(FilaPrioridade));
14
15
        if (filaP != NULL)
            *filaP = NULL;
16
17
        return filaP;
18
    }
19
    NO *alocarNO(){
20
21
        return (NO *)malloc(sizeof(NO));
22
    }
23
    void liberarNO(NO *q){
24
25
        free(q);
26
27
    int estaVazia(FilaPrioridade *filaP){
28
29
        if (filaP == NULL)
30
            return -1;
31
        return ((*filaP)==NULL);
32
33
34
    int tamanho(FilaPrioridade *filaP){
        if (filaP == NULL)
35
36
            return -1;
37
        if (estaVazia(filaP))
38
            return 0;
39
        int tamanho = 0;
40
        NO* auxiliar = *filaP;
        while (auxiliar != NULL){
41
42
            auxiliar = auxiliar->proximo;
43
            tamanho++;
44
45
        return tamanho;
46
    }
47
    int inserirPrio(FilaPrioridade* filaP, int elemento, int prioridade){
48
49
        if (filaP == NULL)
50
            return 0;
        NO* novo = alocarNO();
51
        if (novo == NULL)
52
53
            return 0;
54
        novo->info = elemento;
55
        novo->prioridade = prioridade;
56
        if (estaVazia(filaP)){
57
            novo->proximo = *filaP;
```

```
*filaP = novo;
 58
 59
             NO* auxiliar, *anterior;
 60
             auxiliar = *filaP;
 61
 62
             anterior = NULL;
 63
             while (auxiliar != NULL && auxiliar->prioridade >= novo->prioridade){
 64
                  anterior = auxiliar;
 65
                  auxiliar = auxiliar->proximo;
 66
 67
             if (anterior == NULL){
 68
                 novo->proximo = *filaP;
                 *filaP = novo;
 69
 70
             } else {
 71
                 novo->proximo = anterior->proximo;
 72
                  anterior->proximo = novo;
 73
 74
         }
 75
         return 1;
 76
 77
     int removeInicio(FilaPrioridade* filaP){
 78
 79
         if (filaP == NULL)
 80
             return 0;
         if (estaVazia(filaP))
 81
 82
             return 0;
 83
         NO* auxiliar = *filaP;
         *filaP = auxiliar->proximo;
 85
         liberarNO(auxiliar);
 86
         return 1;
 87
     }
 88
     int verInicio(FilaPrioridade* filaP, int* elemento, int* prioridade){
 89
         if (filaP == NULL)
 90
91
             return 0;
 92
         if (estaVazia(filaP))
93
             return 0;
         *elemento = (*filaP)->info;
 94
 95
         *prioridade = (*filaP).prioridade;
96
         return 1;
 97
     }
98
99
     void imprime (FilaPrioridade* filaP){
         if (filaP == NULL)
100
101
             return;
102
         if (estaVazia(filaP)){
             printf ("Fila de Prioridades Vazia!\n");
103
104
             return;
105
106
         NO* auxiliar = *filaP;
         while (auxiliar != NULL){
107
108
             printf ("[%d, %d] ", auxiliar->prioridade, auxiliar->info);
109
             auxiliar = auxiliar->proximo;
110
111
         printf ("\n");
112
     }
113
     void destroiFila(FilaPrioridade* filaP){
114
         if (filaP != NULL){
115
116
             NO* auxiliar;
117
             while ((*filaP) != NULL){
```

```
13/10/2023, 14:40
```

fila_prioridades_heap_binaria.h

```
1 #ifndef FPHEAP H
 2
    #define FPHEAP_H
 3
    #include <stdio.h>
    #include <stdlib.h>
 5
    #define MAX 100
 6
 7
    typedef struct NO{
 8
        int info, prioridade;
 9
    }NO;
    typedef struct{
10
11
        int qtd;
        NO dados[MAX];
12
    }FilaPrioridade;
13
14
15
    FilaPrioridade* criaFila(){
16
        FilaPrioridade* filaP;
        filaP = (FilaPrioridade*)malloc(sizeof(FilaPrioridade));
17
        if (filaP != NULL)
19
            filaP->qtd = 0;
20
        return filaP;
21
    }
22
23
    void destroiFila(FilaPrioridade* filaP){
        if (filaP != NULL)
24
25
            free(filaP);
26
27
    int tamanho(FilaPrioridade* filaP){
28
29
        if (filaP == NULL)
30
            return -1;
31
        return filaP->qtd;
32
33
34
    int estaCheia(FilaPrioridade* filaP){
35
        if (filaP == NULL)
36
            return -1;
37
        return (filaP->qtd == MAX);
38
39
40
    int estaVazia(FilaPrioridade* filaP){
41
        if (filaP == NULL)
42
            return -1;
        return (filaP->qtd = 0);
43
44
    }
45
46
    void imprime(FilaPrioridade* filaP){
47
        if (filaP == NULL)
48
            return;
49
        if (estaVazia(filaP)){
            printf ("Fila vazia!\n");
50
51
            return;
52
53
        printf ("Elementos: \n");
54
        int i;
55
        for (i=0; i<filaP->qtd; i++)
            printf ("[%d, %d] (%d) -- ", filaP->dados[i].prioridade, filaP->dados[i].info, i);
56
57
        printf ("\n");
```

```
13/10/2023, 14:40
                                                   fila prioridades heap binaria.h
   58
       }
   59
       void trocaNO(NO* a, NO* b){
   60
   61
           NO temp;
   62
           temp.info = a->info;
   63
           temp.prioridade = a->prioridade;
   64
           a->info = b->info;
   65
           a->prioridade = b->prioridade;
           b->info = temp.info;
   66
   67
           b->prioridade = temp.prioridade;
   68
       }
   69
   70
       void ajustaHeapInsere(FilaPrioridade* filaP, int filho){
   71
           NO temp;
           int pai = (filho-1)/2;
   72
   73
           int prioPai = filaP->dados[pai].prioridade;
   74
           int prioFilho = filaP->dados[filho].prioridade;
   75
           while(filho > 0 && prioPai < prioFilho){</pre>
   76
               trocaNO(&filaP->dados[filho], &filaP->dados[pai]);
   77
               filho = pai;
   78
               pai = (pai-1)/2;
   79
               prioPai = filaP->dados[pai].prioridade;
   80
               prioFilho = filaP->dados[filho].prioridade;
   81
           }
  82
       }
   83
       int inserirPrio(FilaPrioridade* filaP, int elem, int pri){
   84
  85
           if(filaP == NULL) return 0;
   86
           if(estaCheia(filaP)) return 0;
   87
           filaP->dados[filaP->qtd].info = elem;
  88
           filaP->dados[filaP->qtd].prioridade = pri;
   89
           ajustaHeapInsere(filaP, filaP->qtd);
  90
           filaP->qtd++;
  91
           return 1;
  92
  93
  94
       void ajustaHeapRemove(FilaPrioridade* filaP, int pai){
  95
           NO temp;
  96
           int filho = 2*pai + 1;
  97
           while(filho < filaP->qtd){
  98
               if(filho < filaP->qtd-1)
  99
                   if(filaP->dados[filho].prioridade < filaP->dados[filho+1].prioridade)
 100
                        filho++;
 101
 102
               if(filaP->dados[pai].prioridade > filaP->dados[filho].prioridade)
 103
 104
               trocaNO(&filaP->dados[pai], &filaP->dados[filho]);
 105
 106
               pai = filho;
               filho = 2*pai + 1;
 107
 108
           }
 109
       }
 110
 111
       int removeInicio(FilaPrioridade* filaP){
           if(filaP == NULL) return 0;
 112
 113
           if(estaVazia(filaP)) return 0;
 114
```

filaP->dados[0].info = filaP->dados[filaP->qtd].info;

filaP->dados[0].prioridade = filaP->dados[filaP->qtd].prioridade;

filaP->qtd--;

115

116117

```
ajustaHeapRemove(filaP, 0);
118
119
        return 1;
120
121
    int verInicio(FilaPrioridade* filaP, int* valor, int* pri){
122
     if(filaP == NULL)
123
124
        return 0;
125
     if(estaVazia(filaP))
        return 0;
126
      *valor = filaP->dados[0].info;
127
      *pri = filaP->dados[0].prioridade;
128
129
     return 1;
130 }
131 #endif //FPHEAP_H
```

1- Criar Fila de Prioridades. Criar Fila de Prioridades. 1- Criar Fila de Prioridades. 2- Inserir item pela prioridade. 2- Inserir item pela prioridade. 2- Inserir item pela prioridade. 3- Ver Inicio. 3- Ver Inicio. 3- Ver Inicio. 4- Remover um Item. 4- Remover um Item. 4- Remover um Item. 5- Imprimir Fila. 5- Imprimir Fila. 5- Imprimir Fila. 6- Mostrar Tamanho. 6- Mostrar Tamanho. 6- Mostrar Tamanho. 7- Destruir Fila. 7- Destruir Fila. 7- Destruir Fila. 8- Sair. 8- Sair. 8- Sair. Opcao: 7 Opcao: 1 Opcao: 5 Fila destruida! [8, 20] [5, 10] 1- Criar Fila de Prioridades. 1- Criar Fila de Prioridades. 2- Inserir item pela prioridade. 1- Criar Fila de Prioridades. 2- Inserir item pela prioridade. 3- Ver Inicio. 2- Inserir item pela prioridade. 3- Ver Inicio. 4- Remover um Item. 3- Ver Inicio. 4- Remover um Item. 5- Imprimir Fila. 4- Remover um Item. 5- Imprimir Fila. 6- Mostrar Tamanho. 5- Imprimir Fila. 6- Mostrar Tamanho. 7- Destruir Fila. 6- Mostrar Tamanho. 7- Destruir Fila. 8- Sair. 7- Destruir Fila. 8- Sair. Opcao: 2 8- Sair. Opcao: 8 Digite o numero que deseja enfileirar: Opcao: 6 Saindo. Tamanho: 2 Prioridade do Elemento: 1- Criar Fila de Prioridades. 2- Inserir item pela prioridade. Enfileirou (10) 3- Ver Inicio. 1- Criar Fila de Prioridades. 4- Remover um Item. Inserir item pela prioridade. 5- Imprimir Fila. 3- Ver Inicio. 6- Mostrar Tamanho. 4- Remover um Item. 7- Destruir Fila. 5- Imprimir Fila. 8- Sair. 6- Mostrar Tamanho. Opcao: 4 7- Destruir Fila. Desenfileirou o primeiro elemento. 8- Sair. 1- Criar Fila de Prioridades. Opcao: 2 Inserir item pela prioridade. Digite o numero que deseja enfileirar: 3- Ver Inicio. 20 4- Remover um Item. Prioridade do Elemento: 5- Imprimir Fila. 6- Mostrar Tamanho. Enfileirou (20) 7- Destruir Fila. 1- Criar Fila de Prioridades. 8- Sair. Inserir item pela prioridade. Opcao: 5 3- Ver Inicio. [5, 10] 4- Remover um Item. 5- Imprimir Fila. 6- Mostrar Tamanho. 7- Destruir Fila. 8- Sair. Opcao: 3 Inicio: 20 (Prioridade: 8)

1- Criar Fila de Prioridades.	1- Criar Fila de Prioridades.
2- Inserir item pela prioridade.	2- Inserir item pela prioridade.
3- Ver Inicio.	3- Ver Inicio.
4- Remover um Item.	4- Remover um Item.
5- Imprimir Fila.	5- Imprimir Fila.
6- Mostrar Tamanho.	6- Mostrar Tamanho.
7- Destruir Fila.	7- Destruir Fila.
8- Sair.	8- Sair.
Opcao: 1	
1- Criar Fila de Prioridades.	Opcao: 5 Elementos:
2- Inserir item pela prioridade.	ETelleticos.
3- Ver Inicio.	1 Crier File de Brieridades
4- Remover um Item.	1- Criar Fila de Prioridades.
5- Imprimir Fila.	2- Inserir item pela prioridade.
6- Mostrar Tamanho.	3- Ver Inicio.
7- Destruir Fila.	4- Remover um Item.
8- Sair.	5- Imprimir Fila.
Opcao: 2	6- Mostrar Tamanho.
Digite o numero que deseja enfileirar:	7- Destruir Fila.
1	8- Sair.
Prioridade do Elemento:	Opcao: 6
10	Tamanho: 0
Enfileirou (1)	1- Criar Fila de Prioridades.
1- Criar Fila de Prioridades.	2- Inserir item pela prioridade.
2- Inserir item pela prioridade.	3- Ver Inicio.
3- Ver Inicio.	4- Remover um Item.
4- Remover um Item.	5- Imprimir Fila.
5- Imprimir Fila.	6- Mostrar Tamanho.
6- Mostrar Tamanho.	7- Destruir Fila.
7- Destruir Fila.	8- Sair.
8- Sair.	Opcao: 7
Opcao: 2	Fila destruida!
Digite o numero que deseja enfileirar:	1- Criar Fila de Prioridades.
2	2- Inserir item pela prioridade.
Prioridade do Elemento:	3- Ver Inicio.
5	4- Remover um Item.
Enfileirou (2)	5- Imprimir Fila.
1- Criar Fila de Prioridades.	6- Mostrar Tamanho.
2- Inserir item pela prioridade.	7- Destruir Fila.
3- Ver Inicio.	8- Sair.
4- Remover um Item.	Opcao: 8
5- Imprimir Fila.	Saindo.
6- Mostrar Tamanho.	
7- Destruir Fila.	
8- Sair.	
Opcao: 3	
Inicio: 1 (Prioridade: 10)	