Programação 2 _ T5x

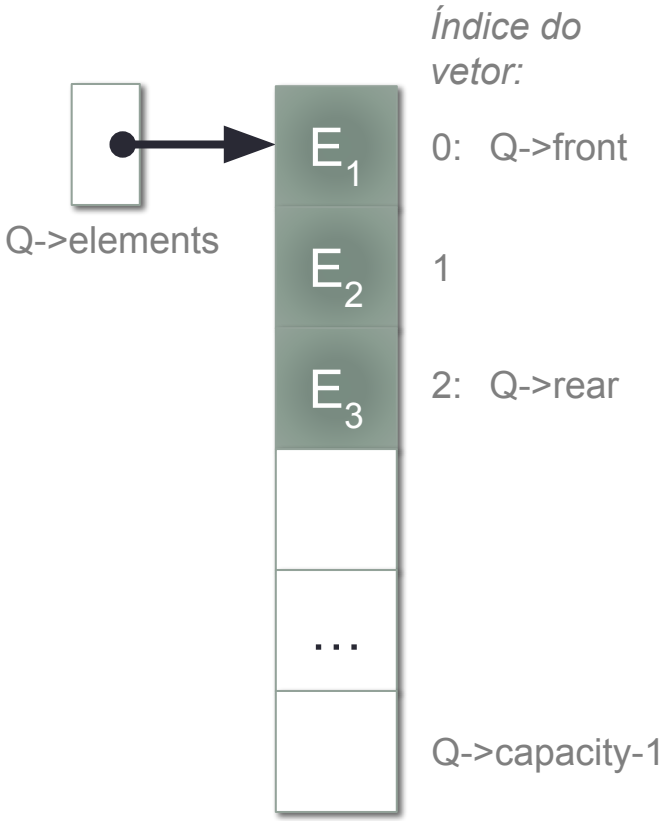
Filas e Pilhas – Implementações baseadas em vetores e listas

Rui Camacho
(slides por Luís Teixeira)
MIEEC 2020/2021

Fila (Implementação baseada em Vetor)

```
struct queueItem
{
    int capacity; /* capacidade da fila */
    int front; /* índice da cabeça da fila
*/
    int rear; /* índice da cauda da fila */
    int size; /* tamanho da fila */
    /* vetor com os elementos */
    data_type *elements;
};
typedef struct queueItem* Queue;
```

```
/* cria uma nova fila */
Queue CreateQueue( int maxSize );

/* insere um novo elemento na cauda*/
void Push( data_type X, Queue Q ) ;

/* remove o elemento da frente */
void Pop( Queue Q );

/* obtém o valor do elemento da frente */
data_type Front( Queue Q );
```



data_type : deverá ser substituído pelo tipo de dados dos elementos a inserir na fila

Fila (Implementação baseada em Vetor)

```
#define MIN_QUEUE_SIZE 5

Queue CreateQueue( int maxSize )
{
    Queue Q;
    if( maxSize < MIN_QUEUE_SIZE )
        printf( "Queue size is too small\n" );

    Q = ( struct queueItem * ) malloc( sizeof( struct queueItem ) );
    if( Q == NULL ) {
        printf( "Out of space!\n" ); exit(EXIT_FAILURE);
    }
    Q->elements = ( int * ) malloc( sizeof( int ) * maxSize );
    if( Q->elements == NULL ) {
        printf( "Out of space!\n" ); exit(EXIT_FAILURE);
    }

    Q->capacity = maxSize;
    Q->size = 0;
    Q->front = 0;
    Q->rear = 0;
    return Q;
}
```

Fila (Implementação baseada em Vetor)

```
void Push( int X, Queue Q )
{
    if( Q->size == Q->capacity ) printf( "Full queue\n" );
    else {
        Q->size++;
        Q->elements[ Q->rear ] = X;
        if ( ++Q->rear == Q->capacity )
            Q->rear = 0;
    }
}

void Pop( Queue Q )
{
    if( Q->size == 0 ) printf( "Empty queue\n" );
    else
    {
        Q->size--;
        if ( ++Q->front == Q->capacity )
            Q->front = 0;
    }
}
```

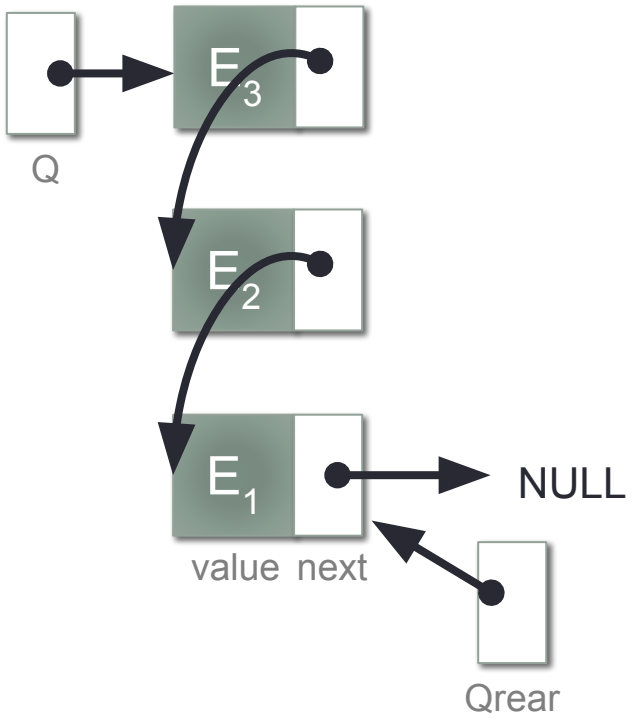
Fila (Implementação baseada em Vetor)

```
int Front( Queue Q )
{
    if( Q->size != 0 )
        return Q->elements[ Q->front ];
    printf( "Empty queue\n" ); return 0;
}

int main( )
{
    Queue Q; int i;

    Q = CreateQueue(15);
    for( i = 0; i < 10; i++ )
        Push( i, Q );

    while( Q->size != 0 )
    {
        printf( "Value: %d\n", Front( Q ) );
        Pop( Q );
    }

    free( Q->elements ); free( Q );
}
```

Qual o é o resultado deste programa?

Fila (Implementação baseada em Lista Ligada)

```
struct queueItem
{
    /* valor do elemento da fila */
    data_type value;
    /* apontador para o elemento seguinte */
    struct queueItem *next;
};
typedef struct queueItem* Queue;
```




```
/* cria uma nova fila */
Queue CreateQueue();

/* insere um novo elemento na cauda */
void Push( data_type X, Queue Q, Queue *Qrear
);

/* remove o elemento da frente */
void Pop( Queue Q );

/* obtém o valor do elemento da frente */
data_type Front( Queue Q );
```

data_type : deverá ser substituído pelo tipo de dados dos elementos a inserir na fila

Fila (Implementação baseada em Lista Ligada)

```
Queue CreateQueue( void )
{
    Queue Q;
    Q = (struct queueItem *) malloc( sizeof( struct queueItem ) );
    if( Q == NULL ) {
        printf("Out of space!!!\n"); exit(EXIT_FAILURE);
    }
    Q->next = NULL;  return Q;
}

void Push( int X, Queue Q, Queue *Qrear )
{
    struct queueItem *newItem;

    newItem = (struct queueItem *) malloc( sizeof( struct queueItem ) );
    if( newItem == NULL ) {
        printf( "Push: Out of space!!!\n" ); exit(EXIT_FAILURE);
    }
    newItem->value = X;
    newItem->next = NULL;

    (*Qrear)->next = newItem;
    *Qrear = newItem;
}
```

Fila (Implementação baseada em Lista Ligada)

```
void Pop( Queue Q )
{
    struct queueItem *tmp;

    if( Q->next == NULL )
        printf( "Empty queue\n" );
    else
    {
        tmp = Q->next;
        Q->next = tmp->next;
        free(tmp);
    }
}

int Front( Queue Q )
{
    if( Q->next != NULL )
        return Q->next->value;
    printf( "Empty queue\n" );
    return -1;
}
```

Fila (Implementação baseada em Lista Ligada)

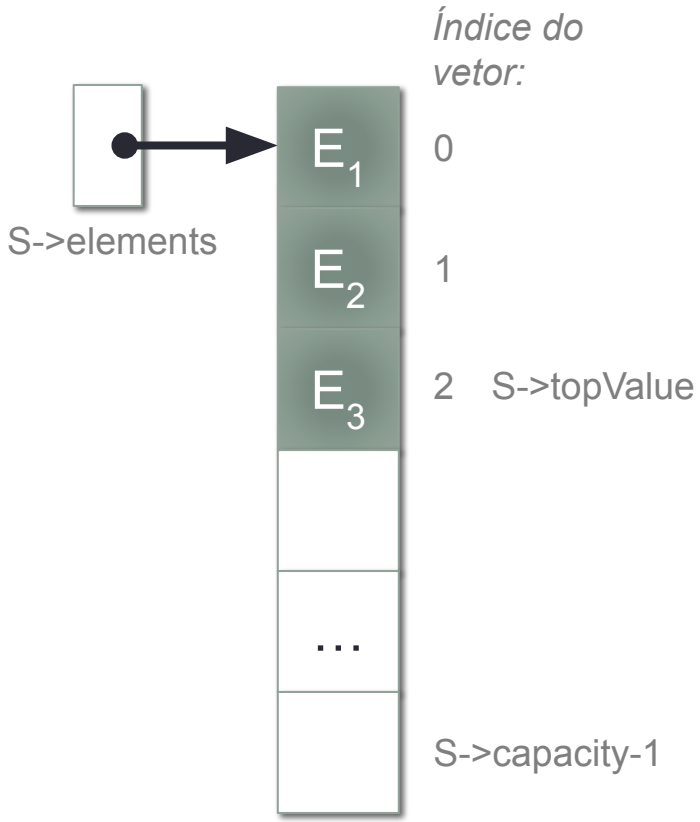
```
int main( )
{
    Queue Q, Qrear;
    int i;

    Q = Qrear = CreateQueue();

    for( i = 0; i < 10; i++ )
        Push( i, Q , &Qrear);

    while( Q->next != NULL )
    {
        printf( "%d\n", Front( Q ) );
        Pop( Q );
    }

    free( Q );
}
```

Qual o é o resultado deste programa?

Pilha (Implementação baseada em Vetor)

```
struct stackItem
{
    /* capacidade da stack */
    int capacity;
    /* índice do topo da stack */
    int topValue;
    /* vetor com os elementos */
    data_type *elements;
};
typedef struct stackItem* Stack;
```

```
/* cria uma nova stack */
Stack CreateStack( int maxSize );

/* insere um novo elemento no topo */
void Push( data_type X, Stack S ) ;

/* remove o elemento do topo */
void Pop( Stack S );

/* obtém o valor do elemento do topo */
data_type Top( Stack S );
```



data_type : deverá ser substituído pelo tipo de dados dos elementos a inserir na fila

Pilha (Implementação baseada em Vetor)

```
#define EMPTY_STACK -1
#define MIN_STACK_SIZE 5

Stack CreateStack( int maxSize )
{
    Stack S;
    if( maxSize < MIN_STACK_SIZE ) {
        printf("Stack size is too small\n" ); exit(EXIT_FAILURE);
    }

    S = (struct stackItem *) malloc( sizeof(struct stackItem) );
    if( S == NULL ) {
        printf( "Out of space!\n" ); exit(EXIT_FAILURE);
    }
    S->elements = (int *) malloc( sizeof(int) * maxSize );
    if( S->elements == NULL ) {
        printf( "Out of space!\n" );
        exit(EXIT_FAILURE);
    }
    S->capacity = maxSize;
    S->topValue = EMPTY_STACK;
    return S;
}
```

Pilha (Implementação baseada em Vetor)

```
void Push( int X, Stack S )
{
    if( S->topValue == S->capacity - 1 ) {
        printf( "Full stack\n" ); exit(EXIT_FAILURE);
    }
    else S->elements[ ++S->topValue ] = X;
}

void Pop( Stack S )
{
    if( S->topValue == EMPTY_STACK )
        printf( "Empty stack\n" );
    else S->topValue--;
}

int Top( Stack S )
{
    if( S->topValue != EMPTY_STACK )
        return S->elements[ S->topValue ];
    printf( "Full stack\n" ); exit(EXIT_FAILURE);
    return 0;
}
```

Pilha (Implementação baseada em Vetor)

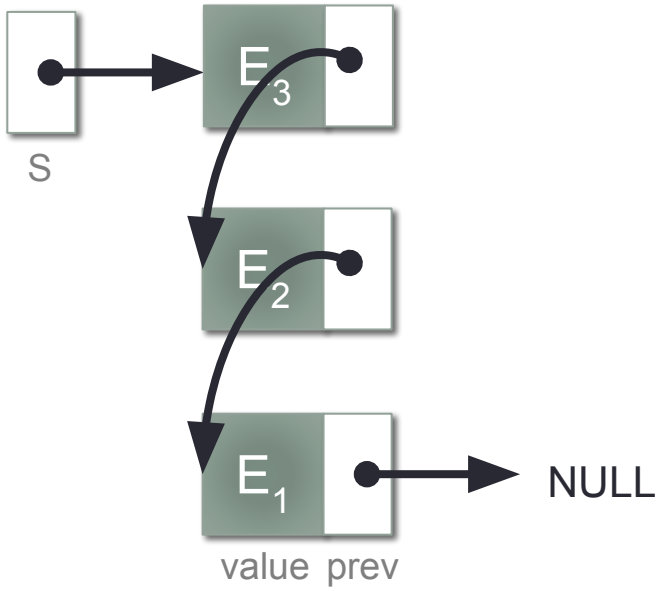
```
int main( )
{
    Stack S;
    int i;

    S = CreateStack( 15 );
    for( i = 0; i < 10; i++ )
        Push( i, S );

    while( S->topValue != EMPTY_STACK )
    {
        printf( "%d\n", Top( S ) );
        Pop( S );
    }

    if( S != NULL ) {
        free( S->elements ); free( S );
    }
}
```

Qual o é o resultado deste programa?

Pilha (Implementação baseada em Lista Ligada)

```
struct stackItem
{
    /* valor do elemento da stack */
    data_type value;
    /* apontador para o elemento anterior
*/
    struct stackItem *prev;
};
typedef struct stackItem* Stack;
```

```
/* cria uma nova stack */
Stack CreateStack();

/* insere um novo elemento no topo */
void Push( data_type X, Stack S ) ;

/* remove o elemento do topo */
void Pop( Stack S );

/* obtém o valor do elemento do topo */
data_type Top( Stack S );
```



data_type : deverá ser substituído pelo tipo de dados dos elementos a inserir na fila

Pilha (Implementação baseada em Lista Ligada)

```
Stack CreateStack( void )
{
    Stack S;
    S = (struct stackItem *) malloc( sizeof( struct stackItem ) );
    if( S == NULL ) {
        printf("Out of space!!!\n"); exit(EXIT_FAILURE);
    }
    S->prev = NULL;
    return S;
}

void Push( int X, Stack S )
{
    struct stackItem *tmp;
    tmp = (struct stackItem *) malloc( sizeof( struct stackItem ) );
    if( tmp == NULL ) {
        printf( "Out of space!!!\n" ); exit(EXIT_FAILURE);
    }
    else {
        tmp->value = X;
        tmp->prev = S->prev;
        S->prev = tmp;
    }
}
```

Pilha (Implementação baseada em Lista Ligada)

```
void Pop( Stack S )
{
    struct stackItem *firstElem;

    if( S->prev == NULL )
        printf( "Empty stack\n" );
    else
    {
        firstElem = S->prev;
        S->prev = S->prev->prev;
        free( firstElem );
    }
}

int Top( Stack S )
{
    if( S->prev != NULL )
        return S->prev->value;
    printf( "Empty stack\n" );
    return -1;
}
```

Pilha (Implementação baseada em Lista Ligada)

```
int main( )
{
    Stack S;
    int i;

    S = CreateStack( );
    for( i = 0; i < 10; i++ )
        Push( i, S );

    while( S->prev != NULL )
    {
        printf( "%d\n", Top( S ) );
        Pop( S );
    }

    free( S );
}
```

Qual o é o resultado deste programa?