Documentação Técnica do Software de Simulação Portuária

Desenvolvedor: João Paulo de Souza

Disciplina: Estrutura de Dados

Curso: Ciência da Computação

Instituto Federal de Minas Gerais – Campus Formiga

14 de Maio de 2018 - Formiga/MG

1 - Problema a ser resolvido:

O problema consiste em simular o andamento de um porto em forma de um modelo de simulação conceitual que aborda operações portuárias de um porto fictício inspirado no Terminal de Puerto Nuevo (Buenos Aires – Argentina), apenas na parte de atracamento dos návios. Esses návios estão em quatro filas, para quatro atracamentos, cada unidade de tempo do software um návio de cada fila é retirado para ser desempilhados seus containers e colocados na área de atracamento destinado a isso, conhecida como travessas. Essas travessas comportam cinco containers cada, então como são cinco travessas, 25 é o máximo de containers possíveis. Então quando enche essas travessas, há a necessidade de um veículo carregador ir e guardar esses containers num local de armazenamento. E assim vai executando para todas as áreas de atracamento.

2-Estruturas/TADs

Foram definidas algumas TADs para auxiliar no processo de desenvolvimento, elas estão distruídas entre as TADs das bibliotecas de pilha(stack) e de fila(queue). Essas bibliotecas tem como função permitir o salvamento dos navios e seus containers e após a vez de cada navio atracar, realizar o empilhamento nas travessas(áreas de empilhamento de containers).

2.1- Estruturas/TADS presentes nas bibliotecas

2.1.1 Biblioteca Queue.h

A biblioteca queue tem a função de simular um tipo abstrato de dados chamado fila, onde o conceito básico é, o primeiro que entra é o primeiro que sai. Essa biblioteca fui fundamental para a criação do software. Nessa biblioteca há funções e dois ponteiros que permitem a criação de variáveis do tipo do TAD, o que facilita na contrução do software. Esses ponteiros são:

- typedef struct QUEUE* queue;
- typedef struct CELL* cell;

E também na biblioteca de filas, as seguinte funções foram implementadas:

- *void create_queue()* - Função que aloca memória para estrutura de fila.
- *queue start_queue_empty(queue)* – Função que coloca os ponteiros de controle de primeiro e último da fila em NULL e inicia o tamanho da fila com 0, recebe por parâmetro uma variavel do tipo TAD fila.
- *bool verify_queue_empty(queue)* – Função que verifica se a fila está vazia verificando se os ponteiros estão em NULL e se o tamanho é igual a 0, recebe por parâmetro uma variável do tipo TAD fila. Obs: O retorno ‘bool’ da função, foi implementado por outra biblioteca que será explicada abaixo.
- *void line_up(queue,int)* – Função que enfileira os elementos numa fila já existente, incrementando seu tamanho. Tem algumas peculiaridades, uma delas é que

necessita da criação de uma célula(um dos tipos que são ponteiros na biblioteca, então é necessário alocar espaço de memória para o tipo célula, para que assim possa ser inserido um valor, outra peculiaridade, mas essa com o tipo célula, é que ela possui apontadores para seu próximo e seu anterior, então a função controla e garente que sempre que seja inserido uma célula, ela aponte para a anterior e a anterior aponte para ela, para que a referência(localização) não seja perdida. E por fim, outro exemplo peculiaridade seria, se for o primeiro elemento da fila o novo inserido, os dois ponteiros de controle apontem para esse elemento e o apontador próximo e anterior desse elemento apontam para NULL, já depois disso é necessário somente que o ultimo aponte para esse novo elemento(célula) e a regra de próximo e anterior citada acima anteriormente continua normalmente. Recebe por parâmetro uma variável do tipo fila e uma variável do tipo inteiro.

- *cell misalign(queue)* – Função que desinfileira os elementos numa fila já existente e vai decrementando o tamanho. Como a função acima, também possui algumas peculiaridades, além de seguir o mesmo esquema de apontadores para próximo e anterior. Uma das peculiaridades seria que quando só resta um elemento na fila e ele for removido, os apontadores de primeiro e ultimo devem novamente apontar para NULL e por fim o tamanho zerado novamente. Outra peculiaridade é que a função retorna um estrutura inteira de célula, nisso onde ela for usada, quem a usa deve lembrar disso e dar free(liberar espaço de memória)na variável do tipo célula que receber a célula removida. E por fim, nela é verificado se a fila já não está vazia, afinal não pode-se remover algo de uma fila já vazia. Recebe uma variável do tipo fila por parâmetro.
- *void drop_queue(queue)* – Função que desmonta a fila, fazendo com que ela seja desalocada da memória para liberação da mesma. Nela os elementos são retirados um a um pela função misalign(função acima) e armazenados numa variavel do tipo célula, e por fim é usado o comando free nessas variáveis para que os espaços de memória dos elementos seja liberado. Por fim quando todos os elementos forem retirados, da free na própria estrutura da fila, assim ela não existe mais na memória. Recebe uma variável do tipo fila por parâmetro.
- *int queue_length(queue)* – Função que retorna o tamanho da fila, recebe uma variável do tipo fila por parâmetro. Acessa dentro da biblioteca de fila o ponteiro para a estrutura/TAD fila e assim consegue acesso a variável de controle de tamanho, então retorna o valor dessa variável.
- *cell search_value_on_queue(queue,int)* – Função que procura o indice de um valor dentro da fila, caso não encontre, retorna NULL, se encontrar, retorna a célula referente a esse elemento. Recebe por parâmetro uma variável do tipo fila e uma varíavel do tipo inteiro.
- *void show_queue(queue)* – Função que mostra todos os elementos da fila, utilizando um laço de repetição(loop) que executa até o tamanho da fila. Recebe por parâmetro uma varíavel do tipo fila.
- *int save_on_other_local(cell)* – Função que pega um valor inteiro presente em uma variável do tipo célula e retorna esse valor. Recebe a variável do tipo célula por parâmetro, pelo acesso global do apontador de célula, acessa o valor contido na célula e retorna esse valor.

- *stack save_on_other_stack(cell,int)* – Função que pega uma pilha(outra TAD, que será explicado futuramente)dentro da estrutura de célula e retorna essa pilha de acordo com uma variável do tipo inteiro que controla o index da pilha a ser retornado, pois na célula contém um vetor da TAD pilha. Retorna então a pilha de acordo com o index pelo acesso global concedido pelo apontador da estrutura célula. Recebe como parâmetro uma varíavel do tipo célula e uma variável do tipo inteiro.
- *void save_cell_on_file(cell,int)* – Função que salva uma célula inteira em um arquivo texto. Recebe por parametro uma variável do tipo célula por parâmetro e uma variável do tipo inteiro. Pega todos os valores da célula, pelo acesso global concedido pelo apontador da estrutura célula e vai salvando em um arquivo texto.

2.1.2 Biblioteca Stack.h

A biblioteca stack tem com função simular um tipo abstrato de dados chamado pilha, que tem como conceito básico, o primeiro que entra é o último que sai. Também é outra biblioteca extremamente útil ao software. Nessa biblioteca também há funções e dois ponteiros, que auxiliam na criação das funções e de váriaveis deste mesmo tipo, afim de utiliza-las no software. Os ponteiros são:

- typedef struct STACK* stack;
- typedef struct CELLULE* cellule;

E as funções são:

- *stack create_stack()* - Função que aloca memória para estrutura da pilha.
- *void start_stack_empty(stack)* - Função que coloca o ponteiro de controle de topo da pilha em NULL e inicia o tamanho da pilha com 0, recebe por parâmetro uma variavel do tipo TAD fila.
- *bool verify_stack_empty(stack)* - Função que verifica se a pilha está vazia verificando se o ponteiro de controle está em NULL e se o tamanho é igual a 0, recebe por parâmetro uma variável do tipo TAD pilha. Obs: O retorno ‘bool’ da função, foi implementado por outra biblioteca que será explicada abaixo.
- *void stack_up(stack,int)* - Função que empilha os elementos numa pilha já existente, incrementando seu tamanho. Tem algumas peculiaridades, uma delas é que necessita da criação de uma célula(um dos tipos que são ponteiros na biblioteca, então é necessário alocar espaço de memória para o tipo célula, para que assim possa ser inserido um valor, outra peculiaridade, mas essa com o tipo célula, é que ela possui apontadores para seu próximo e seu anterior, então a função controla e garente que sempre que seja inserido uma célula, ela aponte para a anterior e a anterior aponte para ela, para que a referência(localização) não seja perdida. Recebe por parâmetro uma variável do tipo pilha e uma variável do tipo inteiro.
- *cellule unstack(stack)* – Função que desempilha os elementos numa pilha já existente e vai decrementando o tamanho. Como a função acima, também possui algumas peculiaridades, além de seguir o mesmo esquema de apontadores para próximo e anterior. Uma das peculiaridades seria que quando só resta um elemento na pilha e ele for removido, o apontador do topo deve novamente apontar

para NULL e por fim o tamanho zerado novamente. Outra peculiaridade é que a função retorna um estrutura inteira de célula, nisso onde ela for usada, quem a usa deve lembrar disso e dar free(liberar espaço de memória)na variável do tipo célula que receber a célula removida. E por fim, nela é verificado se a pilha já não está vazia, afinal não pode-se remover algo de uma pilha já vazia. Recebe uma variável do tipo pilha por parâmetro.

- *int stack_length(stack)* - Função que retorna o tamanho da pilha, recebe uma variável do tipo pilha por parâmetro. Acessa dentro da biblioteca de pilha o ponteiro para a estrutura/TAD pilha e assim consegue acesso a variável de controle de tamanho, então retorna o valor dessa variável.
- *int search_value_on_stack(stack,int)* – Função que um valor inteiro contido na pilha. Recebe por parâmetro uma variável do tipo pilha e uma variável do tipo inteiro. Pelo acesso direto concedido pelo apontador global da estrutura de pilha, acessa-a e pega o valor inteiro de acordo com o índice, controlado pela variável inteira e retorna esse valor.
- *void show_stack(stack)* – Função que mostra todos os elementos presentes na pilha utilizando um laço de repetição(loop) que executa até o tamanho da pilha. Recebe por parâmetro uma varíavel do tipo pilha.
- *int return_int_value_of_stack_cell(cellule)* – Função que retorna um valor inteiro a partir de uma valor do mesmo tipo presente na estrutura de célula, no qual o acesso é permitido pelo apontador global da estrutura. Pega então esse valor inteiro e o retorna. Recebe por parâmetro uma variável do tipo célula.
- *int return_int_value_of_stack(stack,int)* – Função que retorna um valor inteiro a partir de uma pilha e um determinado índice. Recebe por parâmetro uma variável do tipo TAD pilha e uma variável do tipo inteiro. Acessa a pilha pelo apontador global para a própria estrutura e a partir do índice, retorna o valor presente na pilha, caso não encontrar retorna -1. Vai procurando em um laço de repetição que vai até o fundo da pilha.

2.1.3 Biblioteca Bool.h

A biblioteca bool tem como responsabilidade representar o tipo de dados booleano, esse que é nativamente inexistente na linguagem de programação C. Não tem nenhuma função, apenas uma enumeração, que faz essa representação, assimilando o valor true a 1 e o valor false a 0, que na prática vai realizar essa representação de true e false nos comandos condições da linguagem.

2.1.4 Biblioteca Keyboard.h

A biblioteca keyboard tem também apenas uma função, que tem como responsabilidade detectar quando alguma tecla do teclado for pressionada. É a biblioteca mais importante basicamente, sua função controla a parada do software. A função é:

- *bool keyboard_pressed()* – Função que verifica se uma tecla do teclado foi pressionado, e devolve o resultado em verdadeiro ou falso. Funciona a partir de uma flag da linguagem que detecta essa condição.

3 – Estrutura de funcionamento do arquivo principal

O arquivo principal, definido como main, está organizado da seguinte maneira.

Estruturas:

- *typedef struct SHIP ship* – Contendo os seguintes atributos: unsigned int id, stack containers[4], int amout_containers.
- *typedef struct DOCK dock* – Contendo os seguintes atribuitos: unsigned int id, stack platter[5].
- t*ypedef struct CARRIER carrier* – Contendo o seguinte atributo: unsigned int id_container.

Funções:

- *int container_generator()* – Função que cria containers(hipotéticos representados por seu id)e retorna o valor desse id. Utiliza-se da função rand() da biblioteca time.h para gerar um id aleatório para os containers.
- *ship * ship_generator(int amount)* – Função que cria uma quantidade dada pela variável inteira passada por parâmetro de návios hipotéticos representados por seu id, e que contém containers criados pela função acima. Um id é gerado pela função rand() e com um laço de repetição os containers são inseridos no vetor de pilha. Assim que todos forem criados retorna um ponteiro para o vetor de návios criado dentro da função.
- *void chronometer(int seconds)* – Função que conta uma quantidade de segundos pela quantidade passada por parâmetro numa variável inteira. Utiliza-se do clock do computador para pegar a diferênça entre um clock final e um inicial e compara com o valor em segundos passado no parâmetro, até que não seja igual, não sai da função, que roda em um loop semi-infinito.

E por fim função principal no qual a maioria do código do arquivo principal se concentra. No qual executa as seguintes ações:

- Primeiro cria as variáveis necessárias para o funcionamento e controle da aplicação
- Seta a semente para a criação dos números pseudo-aleatórios
- Cria os vetores de tamanho 4 das estruturas acima e de fila.
- Pega a hora local do computador e mostra na tela como horário de inicio do programa
- Cria as filas e pilhas necessárias para o funcionamento do software
- Então entra num laço while que executará até que uma tecla seja pressionada
- Dentro desse laço então, gera uma quantidade de 4 a 16 návios
- Enfileira eles o mais igualmente possível nas filas.
- Após enfileira-los, mostra as quatro filas e seus elementos

- Em seguida é a parte de atracamento dos navios
- Começa pela primeira fila, verifica se ela não está vazia e remove um návio dessa fila, salva esse návio removido no arquivo de relatório e empilha seus containers nas travessas do seu atracamento. Faz isso para todas as filas e atracamentos, retira o návio da fila e coloca no atracamento e desempilha seus containers para serem colocados nas travessas.
- Quando a quantidade de containers nas travessas chega em 25, que é o máximo, é necessário que venha o carrier para descarregar os containers. Assim o atracamento para enquanto o carrier não liberar as travessas e assim que libera continua o fluxo normalmente.
- Os containers retirados pelo carrier, são salvos no local de armazenamento, onde indica que os containers foram armazenados.
- No fim da aplicação, mostra os návios que ainda estão na fila e abre o arquivo de relatório

Essas ações representando o conceito de uma iteração da aplicação, obviamente com mais iterações podem haver algumas poucas alterações que diz respeito a valores ou informações de validação.

4 – Outras considerações

4.1 Makefile

Uma consideração a ser feita é sobre o arquivo Makefile do código, nele as bibliotecas são todas linkadas(conectadas) para que possam ser usadas as funções das mesmas na função principal. Os comandos do arquivo Makefile são:

```
main: main.o queue.o stack.o keyboard.o
	gcc main.o queue.o stack.o keyboard.o -o exe
	rm *.o
main.o: main.c
	gcc -c main.c
stack.o: stack.h stack.c
	gcc -c stack.c
queue.o: queue.h queue.c
	gcc -c queue.c
keyboard.o: keyboard.c keyboard.h
	gcc -c keyboard.c
```

Onde os arquivos objeto(arquivos .o) são os guias para o executável da aplicação, já o comando gcc, é para a compilação da aplicação.

4.2 Conclusão sobre a atividade

E por fim a conclusão sobre a opinião do desenvolvedor sobre atividade. Foi algo interessante de ser desenvolvido, foi necessário aplicar alguns conhecimentos a mais para deixar a resolução melhor, ou seja implica em um melhor aprendizado para quem o desenvolve.