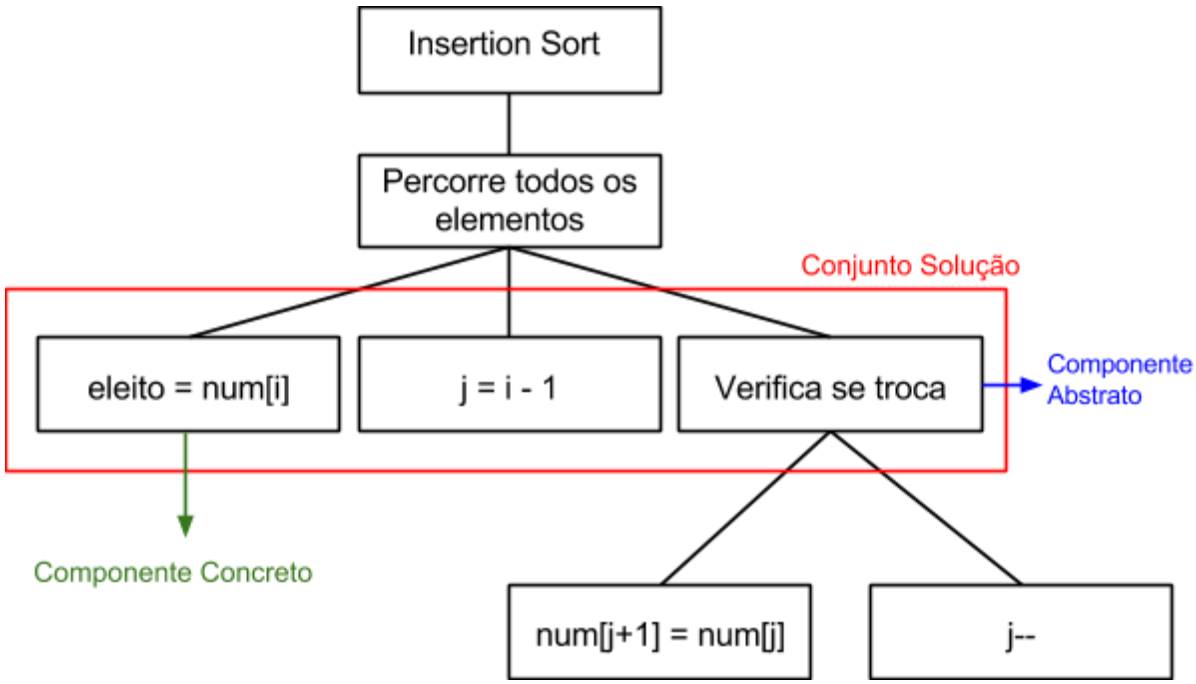
Programação Modular

Lista de Exercícios para a P2

Grupo: Eric Grinstein, Jordana Mecler e Leonardo Wajnzstok

1) **Gere a estrutura de decomposição sucessiva do algoritmo Insertion Sort. Aponte nesta estrutura um componente concreto, um componente abstrato e um conjunto solução.**

```
void insertionSort(int numeros[], int tam) {
  int i, j, eleito;
  for (i = 1; i < tam; i++){
        eleito = numeros[i];
        j = i - 1;
        while ((j>=0) && (eleito < numeros[j])) {
              numeros[j+1] = numeros[j];
              j--;
        }
        numeros[j+1] = eleito;
  }
}
```



2) Faça a argumentação de corretude completa do algoritmo Insertion Sort.

AE → existe um vetor preenchido até a posição tam-1, e tam>=0.
AS → o vetor final estará ordenado em ordem crescente.
AINV → existem 2 conjuntos:

- a ordenar.
- já ordenados.

i aponta para o elemento do conjunto a ordenar.

1a repetição:

1) AE → AINV : pela AE, i aponta para o segundo elemento do vetor (conjunto a ordenar). Neste caso, o primeiro elemento já começa no conjunto ordenado e todos os outros elementos estão no conjunto a ordenar. Vale AINV pois existem 2 conjuntos e i está posicionado corretamente.
2) AE && (C==F) → AS : pela AE, i=1. Como (C==F), tam<=1 e i=1, indicando vetor com apenas 1 posição. Neste caso vale a AS.
3) AE && (C==T) + B → AINV : pela AE, i aponta para o segundo elemento do vetor. Este passa do a ordenar para já ordenados e o i é reposicionado, valendo AINV.
4) AINV && (C==T) + B → AINV : para que AINV continue valendo, B deve garantir que um elemento passe de um conjunto a ordenar para já ordenados e i seja reposicionado.
5) AINV && (C==F) → AS : como (C==F), i é igual ao tam, valendo a AS.
6) Término : O conjunto a ordenar contém um número finito de elementos, e a cada ciclo um deles passa para conjunto já ordenados. A repetição então termina após um número finito de passos.

AI1 → o elemento do vetor na posição i foi atribuído ao eleito.
AI2 → j é posicionado no último ordenado.
AI3 → o elemento se encontra na posição correta.
AS → O eleito foi atribuído ao vetor na posição i.

2a repetição:
AINV2 → existem 2 conjuntos:

- maiores que o elemento
- possíveis menores

j aponta para o elemento a ordenar.

1) AE → AINV2 : pela AE, j aponta para i-1 elemento do vetor (conjunto maiores). Neste caso, todos os elementos do subvetor estão neste conjunto e o conjunto possíveis menores encontra-se vazio. Vale AINV2 pois existem 2 conjuntos e j está posicionado corretamente.
2) AE & (C==F) → AS : pela AE, j=i-1. Como (C==F), eleito>=num[j]. Neste caso, vale a AS.
3) AE && (C==T) + B → AINV2 : pela AE, j aponta para i-1 elemento do vetor. Este passa do conjunto maiores para possíveis menores e j é reposicionado, valendo AINV2.

4) AINV2 && (C==T) + B → AINV2 : para que AINV2 continue valendo, B deve garantir que um elemento passe do conjunto maiores para possíveis menores e j seja reposicionado.
5) AINV2 && (C==F) → AS : com (C==F), j<0 ou eleito>=num[j], valendo AS.
6) Término : o conjunto possíveis menores contém um número finito de elementos e a cada ciclo um deles passa para o outro conjunto. A repetição então termina após um número finito de passos.

3) Faça a argumentação de corretude completa do algoritmo de pesquisa binária.

AE → existe um vetor e este possui tamanho tam. Há uma chave a ser procurada. tam>=0.
AS → retorna a posição da chave no vetor, se achou, ou -1 se não achou.
AI1 → inf aponta para a primeira ocorrência do vetor. Sup aponta para a última ocorrência do vetor.
AI3 → retorna a posição da chave no vetor ou inf>sup.

repetição:
AINV → existem 2 conjuntos:
- a pesquisar.
- já pesquisados.

sup ou inv é reposicionado.
AE → AI1
AS → AI3

1) AE → AINV : pela AE, sup aponta para a última ocorrência do vetor, e inf para a primeira. Vale a AINV, pois existem dois conjuntos e inf e sup estão posicionados corretamente.
2) AE && (C==F) → AS : pela AE, inf aponta para a primeira ocorrência do vetor, e sup para a última. Como (C==F), inf>sup, indicando vetor com tamanho 0. Neste caso, vale a AS.
3) AE && (C==T) + B → AINV : pela AE, inf aponta para a primeira ocorrência do vetor, e sup para a última. O meio passa para o conjunto já pesquisados e inf ou sup é reposicionado, valendo AINV.
4) AINV && (C==T) + B → AINV : para que AINV continue valendo, B deve garantir que um elemento seja passado do conjunto a pesquisar para já pesquisado, e inf ou sup deve ser reposicionado.
5) AINV && (C==F) → AS : com (C==F), sup>inf, valendo a AS.
6) Término : o conjunto a pesquisar possui um número finito de elementos, e a cada ciclo um deles passa para o conjunto já pesquisados. A repetição termina então após um número finito de passos.

AI2 → meio é posicionado para a média entre o sup e o inf.

1a seleção:
AE → AI2

AS → retorna a posição da chave, ou reposiciona inf ou sup.

1) AE && (C==T) + B1 → AS : pela AE, o meio está posicionado. Como (C==T), a chave foi encontrada. Neste caso, B1 retorna a posição da chave, valendo a AS.
2) AE && (C==F) + B2 → AS : pela AE, o meio está posicionado. Como (C==F), a chave ainda não foi encontrada. Neste caso, B2 reposiciona ou inf, ou sup, valendo a AS.

2a seleção:

AE → AI2

AS → retorna a posição da chave, ou reposiciona inf ou sup.

1) AE && (C==T) + B1 → AS : pela AE, o meio está posicionado. Como (C==T), a chave é menor do que o elemento encontrado. Neste caso, B1 reposiciona sup, valendo a AS.
2) AE && (C==F) + B2 → AS : pela AE, o meio está posicionado. Como (C==F), a chave é maior do que o elemento encontrado. Neste caso, B1 reposiciona inf, valendo a AS.

4) Elabore o código de um verificador de lista duplamente encadeada com nó cabeça que armazena em seus nós valores de 0 a 9.

```
LIS_tpCondRet verifica ( LIS_tppLista pLista , int *numErros )
{
	(*numErros) = 0;
	if( pLista->pCorr->pProx != NULL)
	{
		if( pLista->pCorr->pProx->pAnt != pLista->pCorr )
		{
			CNT_Contar(“erro-anterior-do-prox-dif-corr”);
			(*numErros)++;
		}
		else
		{
			CNT_Contar(“ok-anterior-do-prox-igual-corr”);
		}
	}
	if( pLista->pCorr->pAnt != NULL )
	{
		if( pLista->pCorr->pAnt->pProx != pLista->pCorr )
		{
			CNT_Contar(“erro-prox-do-anterior-dif-corr”);
			(*numErros)++;
		}
		else
		{
			CNT_Contar(“ok-prox-do-anterior-igual-corr”);
```

```
                }
        }
        pLista->pCorr = pLista->pOrigem;
        while( pLista->pCorr->prox != NULL )
        {
                if( pLista->pCorr < 0 || pLista->pCorr > 9 )
                {
                        CNT_Contar("erro-valor-invalido");
                        (*numErros)++;
                }
                else
                {
                        CNT_Contar("ok-valor-valido");
                }
        }
        return LIS_CondRetOK;
}
```

5) Elabore um deturpador completo que teste o verificador da questão anterior.

```
LIS_tpCondRet deturpa( LIS_tppLista pLista , int acao )
{
        if( acao == 1 )
        {
                pLista->pCorr->pProx->pAnt == NULL;
        }
        else if( acao == 2 )
        {
                pLista->pCorr->pAnt->pProx == NULL;
        }
        else if( acao == 3 )
        {
                pLista->pCorr = 10;
        }
        return LIS_CondRetOK;
}
```

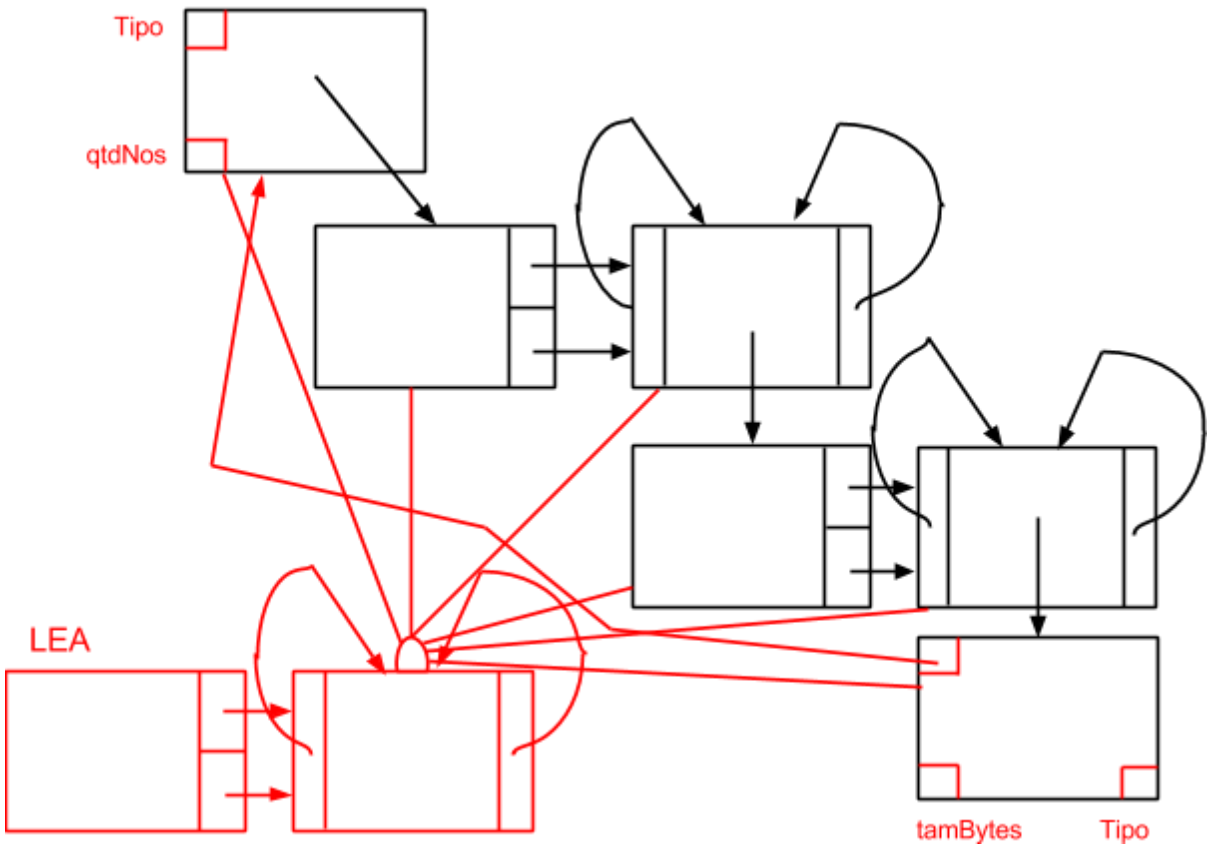
6) Distribua controladores de cobertura no verificador da questão 4 (menor número possível) de forma que contabilize cada entrada e não entrada de seleção.

Já feita na 4).

7) **Gere o script de teste com essas deturpações e verificações.**

==Inicializar contadores
=inicializarcontadores “”
=lercontadores “Contadores”
=iniciarcontagem
==Criar, inserir e deturpar
=criar 0 0
=inserir 1 0
=inserir 3 0
=inserir 7 0
=deturpar 0 3
=verificar 0 1
=voltacorrente 1 0
=deturpar 0 2
=verificar 0 2
=deturpar 0 1
=verificar 0 3

8) **Transforme uma estrutura matriz (lista de listas) que utilize o módulo lista genérica em autoverificável.**

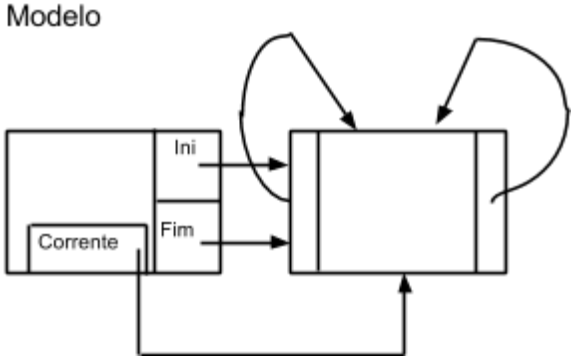


9) Explique como seria implementada uma estrutura de lista duplamente encadeada tolerante a falhas.

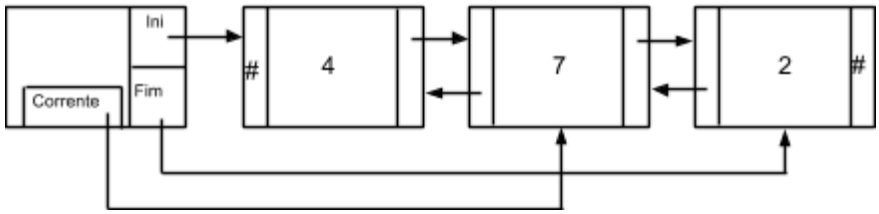
Uma estrutura de lista duplamente encadeada tolerante a falhas seria robusta, ou seja, interceptaria a execução quando observasse um problema, e possuiria mecanismos de recuperação de erros. Por exemplo, uma estrutura de lista encadeada que armazene em seus nós valores de 0 a 9. A estrutura poderia aceitar a inserção de um nó inválido, e ao verificar que o nó é inválido, recuperar o erro reencadeando os nós e excluindo o mesmo.

10) Apresente o modelo e exemplo da estrutura da questão 9.

Modelo



Exemplo

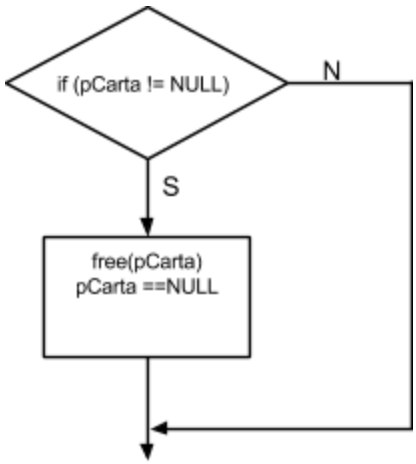


11) Qual é o arrasto de uma repetição existente no algoritmo que gera a seguinte sequência? 1,1,1,3,5,9,17…

3. Percebe-se que os primeiros três inteiros da sequência são inicializados pelo usuário (1,1 e 1). Após isso, o próximo elemento da sequência é calculado como a soma dos três anteriores. O elemento 3 será calculado por 3 números inicializados, o elemento 5 será calculado por 2 números inicializados e um calculado pelo algoritmo (3), o elemento 9 será calculado por 1 número inicializado e 2 calculados pelo algoritmo (3 e 5). Os elementos 17 em diante serão calculados pelo algoritmo. O arrasto é, portanto, 3.

12) Apresente os casos de teste para uma função de acesso do seu trabalho (1,2, ou 3) seguindo o critério caixa aberta.

```
CAR_tpCondRet CAR_ExcluirCarta( CAR_tppCarta pCarta )
{
        if( pCarta != NULL)
        {
                free(pCarta);
                pCarta = NULL;
        }
        return CAR_CondRetOK;
}
```



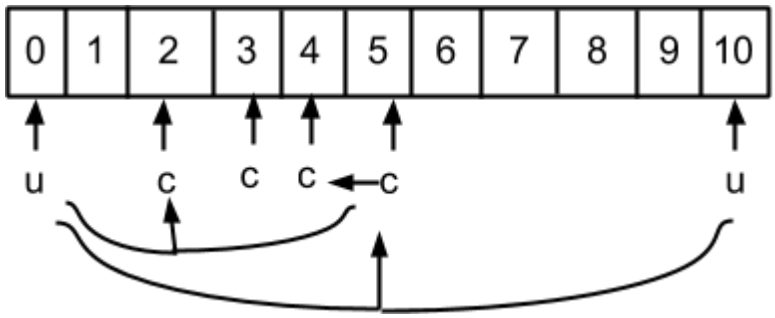
13) Apresente os casos de teste da função da questão 12 pelo padrão de completeza “cobertura de comandos”.

1	pCarta != NULL	S

Casos	1
1	S

14) Explique o valor e como se calcula o arrasto da repetição contida no algoritmo de pesquisa binária.

O arrasto é 2. O início e o fim são inicializados. O meio é computado em cima do início e do fim. Caso o valor buscado seja menor que o meio, o novo fim é computado a partir do meio. O meio é computado a partir do início de do novo fim. Caso o valor buscado seja maior que o meio, o novo início é computado a partir do meio. O meio é computado a partir do novo início e do novo fim.



15) Apresente pelo método de partição em classe de equivalência o script de teste da função excluir_no_corrente da lista duplamente encadeada com nó cabeça.

1a etapa:

grupos:

- lista vazia
- lista com 1 elemento
- lista com 3 elementos

resultados esperados:

- não excluiu
- excluiu e não reencadeou
- excluiu e reencadeou

2a etapa:

Caso	Conjunto	Corrente	Não excluiu	Excluiu
1	vazio		X	
2	A	A		X
3	ABC	A		X
4	ABC	B		X
5	ABC	C		X

3a etapa:

==Caso 1
=cria_lista 0
=exclui_corrente 1

==Caso 2
=insere_elem A 0
=exclui_corrente 0

==Caso 3
=insere_elem A 0
=insere_elem B 0
=insere_elem C 0
=volta_corrente 2 0
=exclui_corrente 0
=destroi_lista 0

==Caso 4
=cria_lista 0
=insere_elem A 0
=insere_elem B 0
=insere_elem C 0
=volta_corrente 1 0
=exclui_corrente 0
=destroi_lista 0

==Caso 5
=cria_lista 0
=insere_elem A 0
=insere_elem B 0
=insere_elem C 0
=exclui_corrente 0
=destroi_lista 0