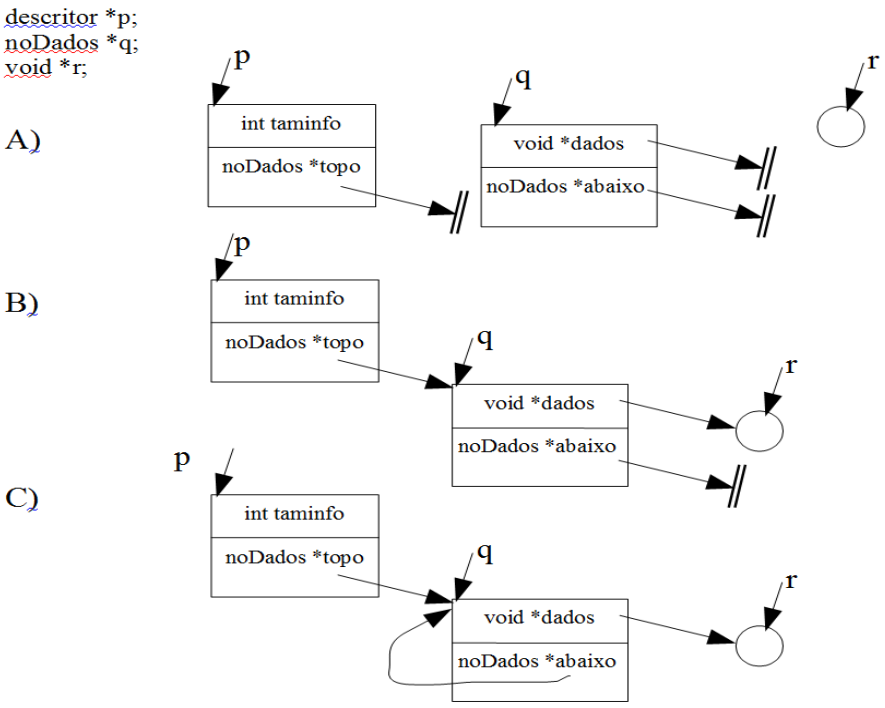
TDA-PILHA

Exercícios

1. Escreva os comandos em linguagem C que levem: (i) da situação 'A' para a 'B' e (ii) da situação 'A' para a 'C', conforme está ilustrado abaixo, onde: *taminfo*, *dados*, *topo* e *abaixo* são campos de structs já criadas e representadas graficamente e o símbolo '→' indica um local referenciado por um apontador.



2. Para a 1 e 2, escreva os comandos em linguagem *C* que, para cada caso, levem do estado 1 para o estado 2:

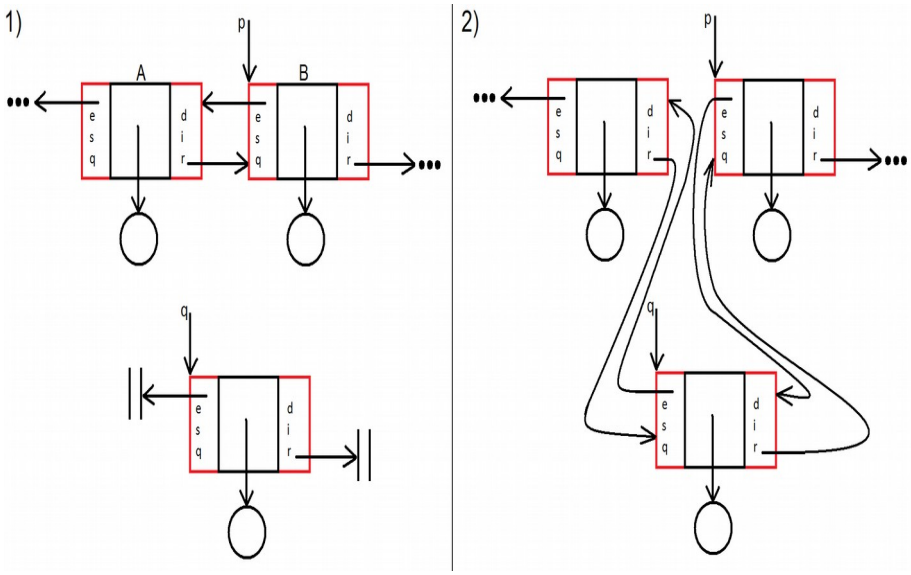


Figura 1: Inserção de elemento duplamente encadeado.

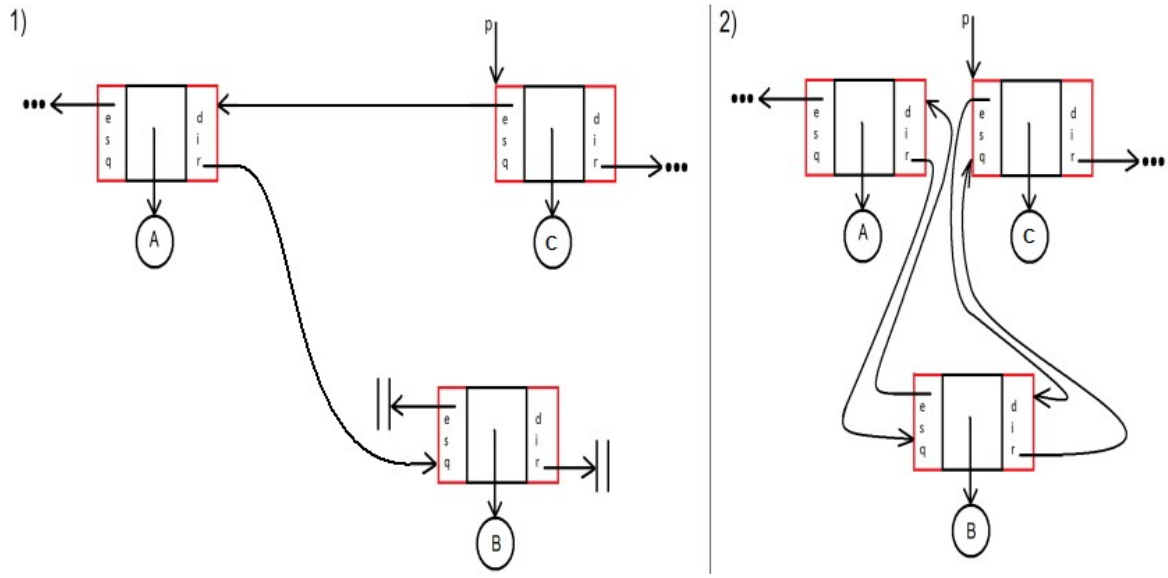


Figura 2: Inserção de elemento duplamente encadeado.

3. Implemente uma aplicação para as operações dos TDAs PE, PDSE e PDDE vistos.

4. Considere uma pilha estática referenciada pelo apontador "p". Atribua valores às variáveis “b” e “c”, abaixo, de maneira que a função “memcpy” copie a informação atualmente no topo da pilha para o endereço em "pDestino". Lembre-se:
   "p->vet[k]" (onde $0 \leq k \leq tamVet-1$) acessa uma posição do vetor e cada posição guarda o respectivo endereço do slot de dados associado.

   memcpy(pDestino, b, c)

   b = _____

   c = _____

5. Construa uma operação (hipotética) int buscaNaBase(Pilha *p, void *pReg) que permite acesso à informação na base da pilha.

6. Com base na pilha dinâmica simplesmente encadeada, implemente um TDA-Pilha Dinâmica Duplamente Encadeada.

7. Implemente a função *int inverte(descritorPDDE *p)* a qual inverte a ordem dos itens em uma PDDE de maneira otimizada, com o menor número de passos. Considerando uma pilha contendo "a, ..., y, z, w", ao final da inversão teremos "w, z, y, ...,a", ou seja, o último que entrou será posicionado no topo da pilha, tornando-se o primeiro a sair, o penúltimo será o segundo a sair e assim sucessivamente até o primeiro que entrou, o qual será o último a sair. Veja a ilustração de uma PDDE ao lado. Não será aceita a solução que simplesmente move o topo para a base da pilha, pois isso não mantém a consistência com as demais operações do TDA (os links usados para o encadeamento – “acima” e “abaixo” – ficariam inconsistentes).

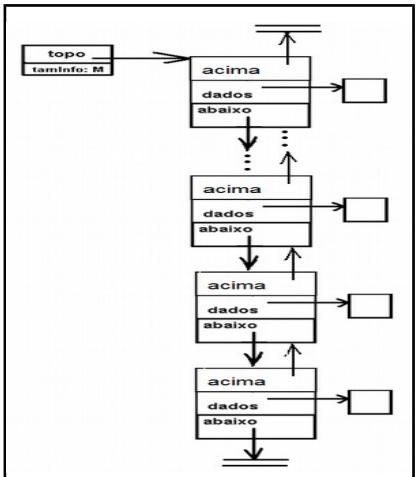


Figura 3: PDDE não vazia.

8. Para uma pilha dinâmica, construa uma função que conta a quantidade de *nós* de dados. Ao final a pilha deve preservar o mesmo estado de antes da execução da função.

9. Discorra sobre a aplicação de pilhas para o rastreio de chamadas recursivas a uma função. A pilha deverá guardar os parâmetros importantes das chamadas.

10. O descritor da PE não anota explicitamente o número atual de itens inseridos. Implemente uma função eficiente (com o menor número de passos) que retorna (return) o número de elementos atualmente empilhados na Pilha Estática. O estado da PE deve ser preservado. Protótipo: *int numeroDeEmpilhados(PE *p)*

11. Implemente a multi-pilha estática (MPE) cujas partições de descritores e de pilhas compartilham o mesmo vetor (utilizando *union*).

12. Considere uma multi-pilha estática contendo duas pilhas em um mesmo vetor:
    I) Inicialização (durante a criação) dos topos das pilhas como $topo_1$ = -1 e $topo_2$ = tamVetor;
    II) Todas as alocações de memória são realizadas na criação;
    III) As pilhas concorrem livremente pelo espaço no vetor, crescendo em direções opostas.
    Implemente as seguintes operações para esse TDA: empilhamento/desempilhamento de uma pilha, teste da condição de uma estar pilha cheia (vazia), destruição, reinicialização da MpE ou de alguma pilha individual, busca em um topo, cálculo do número de elementos inseridos em uma pilha.

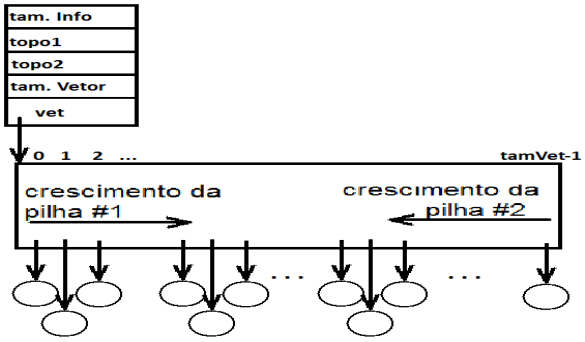


Figura 4: Duas pilhas concorrentes.

13. Dada uma multi-pilha estática contendo várias pilhas sob um mesmo vetor, o qual apresenta um conjunto de células reservadas para atender a uma pilha P qualquer que tenha ficado cheia e não possa atender a uma inserção corrente. Nesse caso, a pilha P poderá utilizar a região de reserva, caso a mesma não tenha sido reclamada anteriormente. As demais pilhas poderão ser deslocadas para benefício da pilha reclamante. Os descritores das pilhas apresentam os respectivos IPs e o FPs anotados. Implemente as seguintes operações: empilhamento/desempilhamento de uma pilha, teste da condição de uma estar pilha cheia (vazia), destruição, reinicialização da MpE ou de alguma pilha individual, busca em um topo, cálculo do número de elementos inseridos em uma pilha.

14. Implemente a função *buscaNoTopoDeUmaPilha(...)*, para o TDA multipilha exibida na 5, a qual retorna SUCESSO ou FRACASSO e copia o conteúdo do elementoa partir do endereço em *pNovoReg* insere em uma pilha *idPilha.*

    Protótipo: *int buscaNoTopoDeUmaPilha(MultiPilha *p, int idPilha, void *destino);*

    ATENÇÃO:
    a) L é o comprimento máximo da partição (iniPart) de uma pilha p_i, $1 \leq i \leq N$;
    b) O início de cada partição *iniPart* consta no descritor individual de cada pilha;
    c) $-1 \leq topo_i \leq (L-1)$;
    d) $topo_i = -1$ indica p_i vazia;
    e) Cada descritor individual guarda o seu *tamInfo*, pois os dados podem variar de tamanho a cada pilha. Os campos foram inicializados na criação desse TDA.

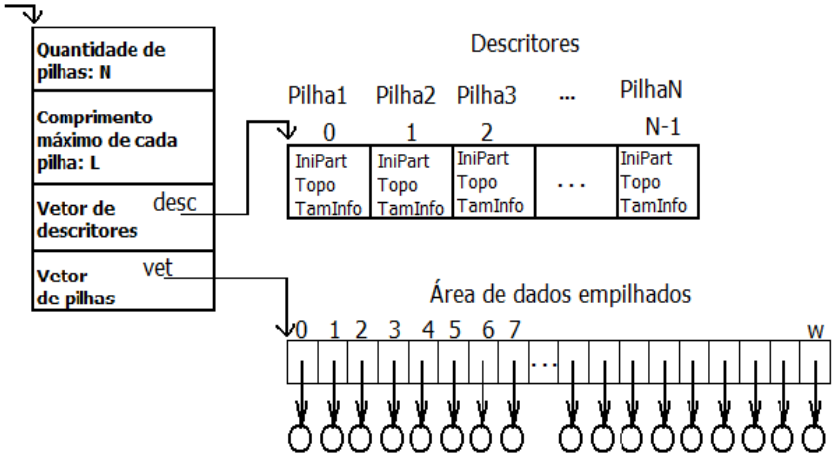


Figura 5: Uma MPE: Descritor geral apontando para vetor de descritores e vetor de dados.

15. Implemente a mesma função da questão 14 considerando que o tamanho de partição (L) varia para cada pilha. Portanto, L tem que constar como atributo em cada descritor individual, não mais fazendo parte do descritor geral.

    *int buscaNoTopoDeUmaPilha(MultiPilha *p, int idPilha, void *destino);*

16. Proponha e implemente uma multi-pilha dinâmica com as seguintes operações: empilhamento/desempilhamento de uma pilha, teste da condição de uma estar pilha cheia (vazia), destruição, reinicialização da MpE ou de alguma pilha individual, busca em um topo, cálculo do número de elementos inseridos em uma pilha.

17. Implemente um novo TDA pilha dinâmica simplesmente encadeada (PDSE) cujo descritor não contém o tamanho da informação (6).

    Será necessário modificar a descrição do nó de dados que deverá guardar o tamanho da informação por ele referenciada. Dessa forma será possível armazenar informações de tamanhos (formatos) diferentes tornando a pilha mais genérica.

    Todas as operações deverão ser modificadas, especialmente a operação de inserção a qual deverá receber o tamanho (em bytes) do item que será inserido. Ao lado, segue uma figura ilustrando a PDSE nas condições descritas.

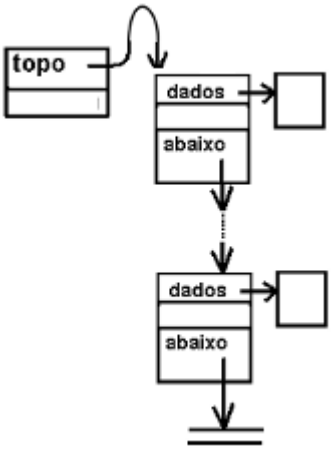


Figura 6: PDSE sem tamInfo no descritor.

18. Justifique suas respostas quanto aos seguintes casos para uma PDSE:

a) Descreva com ilustrações claras o comportamento da função *xxx(...)*, abaixo, considerando sua chamada para uma PDSE nas seguintes condições:

(i) PDSE Vazia (*p->topo==NULL*);

(ii) PDSE Não vazia (com *n* elementos, *n > 1* e *p->topo !==NULL*) cuja inserção ocorreu por meio da função de empilhamento abaixo.

b) Qual o erro na função *xxx(...)* que a leva a uma contagem infinita? Como corrigir isso?

```
int empilha(PDSE *p, void *novo)
{  NoPDSE *temp;
   int ret = FRACASSO;
   if((temp=(NoPDSE *)malloc(sizeof(NoPDSE)))!=NULL)
   { if((temp->dados = (void *) malloc(p->tamInfo)) !=NULL )
     {      memcpy(temp->dados,novo,p->tamInfo);
      if(p->topo==NULL)
            temp->abaixo=temp;
      else
            temp->abaixo=p->topo;
            p->topo=temp;
            ret = SUCESSO;
     }
     else
        free(temp);
   }
   return ret;
}
int xxx(PDSE *p)
{     pNoPDSE aux=NULL;
      int cont =0;
      aux=p->topo)
      while(aux != NULL)
      {     cont++;
            aux=aux->abaixo;
      }
      return cont;
}
```

19. Descreva o funcionamento (teste de mesa) e resultado da execução do código abaixo:

```
int xxx(descritor *p)
{          if(p->topo == NULL)
             return FRACASSO;
           else
          {  tmp=yyy(p, p->topo);
                tmp->abaixo=NULL;
             }
}
```

```
NoPDSE * yyy(descritor *p, NoPDSE *pNo)
{  if (pNo->abaixo==NULL)
          {  p->topo=pNo;
             return pNo;
          }
          else{  aux=yyy(p, pNo->abaixo);
                 aux->abaixo=pNo;
                 return(pNo);
               }
}
```

20. Utilize uma ou mais pilhas para serem aplicadas nos seguintes casos:

a) cálculo de expressões matemáticas pós-fixas. Exemplo: (*A B C ^ / D E * + A C * –*) equivalente à notação infixa (*A / B ^ C + D * E – A * C*), onde o operador x^y corresponde a x^y. (veja solução em HOROWITZ [6]).

Operador	Ordem de precedência
^	3
*, /	2
+,-	1

b) Avaliação do correto “aninhamento” de expressões delimitadas por parênteses, colchetes e chaves. Ex: ([x^{z+y}-t]*[z+w]/x).

c) Ordenação de um vetor com o auxílio de duas pilhas. Este não é um método eficiente para ordenação.

d) Construção de uma função de inserção em ordem em um vetor de inteiros, com auxílio de uma pilha.

e) Conversão de número inteiro decimal em seu equivalente binário, octal ou hexadecimal.

f) Conversão de expressões infixas em pós-fixas (veja solução em HOROWITZ [6]).

g)Adaptado da maratona de programação, treinamento disponível no URL: http://br.spoj.com/problems/ACIDO/:

Um novo RNA foi descoberto. Ele também é constituído de uma cadeia composta de vários intervalos chamados *trans-acting siRNAs*, abreviadamente conhecidos como TASs.

Cada TAS é composto de bases identificadas como B, C, F e S e podem ser pareadas com as bases correspondentes de outro TAS, sendo que os únicos pares possíveis são formados entre as bases B-S e C-F (de TASs diferentes).

O RNA é capaz de dobrar-se sobre si mesmo em intervalos (TASs), para maximizar o número de ligações entre as bases de TASs diferentes. Ao dobrar-se, todas as bases no intervalo se ligam com as suas correspondentes, formando os pares válidos entre bases.

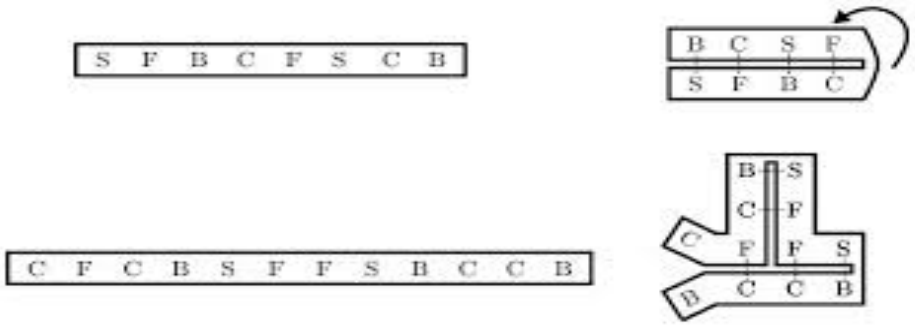


Figura 7: Dobraduras e ligações de bases entre cadeias de TAs.

Considerando que:

- As bases são apenas as quatro seguintes: B, C, F e S
- Cada base pode se ligar a apenas uma outra base (B liga com S e F com C) de outro TAS;
- Não há ordem no emparelhamento, a ligação S-B é tão válida quanto a ligação B-S, o mesmo para C-F e F-C;
- As dobraduras devem ser orientadas para a maximização do número de ligações entre bases;

Utilize uma pilha para determinar o número de ligações entre as bases de uma cadeia ativa.

Exemplos:

Entradas: cadeias	SBC	FCC	SFBC	SFBCFSCB	CFCBSFFSBCCB
Saídas: número de ligações	1	1	0	4	5

h) Um baralho contém 52 cartas distribuídas em quatro naipes: espadas (spades), paus (clubs), ouros (diamonds) e copas (harts). Cada naipe com treze cartas: *1* (*ás), 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 (valete), 12 (dama)* e *13 (rei).*
Considerando a existência do TDA pilha estática, já implementado, cuja interface oferece as funções exibidas na 1, utilizando **apenas** pilhas estáticas e na **menor** quantidade possível, pede-se as seguintes aplicações para o TDA-PE:

i. Construa (em *C*) a estrutura da informação que representa a carta do baralho;
ii. Construa o procedimento *PE* embaralha(PE* pBaralhoCartas)* o qual tem como entrada uma PE apontada por *pBaralhoCartas* e retorna esta mesma pilha embaralhada. Uma ideia é dividir o baralho ao meio e intercalar as cartas para prover alguma desordenação;
iii. Um procedimento comum é separar as 52 cartas nas 4 sequências (correspondentes aos respectivos naipes), sendo cada sequência ordenada pelos valores das cartas:
1 (*ás), 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 (valete), 12 (dama)* e *13 (rei)*. Programe o processo de separação das cartas nos quatro naipes ordenados. Utilize o protótipo abaixo:

void separa(PE pBaralhoCartas, PE* pPEcopas,*
PE pPEpaus, PE* pPEouros, PE* pPEespadas);*

Onde o endereço em *pBaralhoCartas* corresponde à PE que representa o baralho de entrada. Os apontadores *pPEcopas, pPEpaus, pPEouros, pPEespadas* referenciam as PEs de saída, as quais representam as quatro sequências desejadas.

Criação da PE:	`int cria(ppPE pp, int tamPilha, int tamInfo);`
Destruição de qualquer traço da PE na memória:	`int destroi(ppPE pp);`
Força a PE ao estado de recém criada (vazia):	`int reinicia(pPE p);`
Copia a informação de topo no endereço apontado por pRegBuscado:	`int buscaNoTopo(pPE p, void *pRegBuscado);`
Copia a informação de topo no endereço apontado por pRegExcluido e desempilha atualizando o topo:	`int desempilha(pPE p, void *pRegExcluido);`
Empilha copiando o conteúdo apontado por pRegNovo para o novo topo da PE:	`int empilha(pPE p, void *pRegNovo);`
Testa a condição de PE vazia:	`int testaVazia(pPE p);`
Testa a condição de PE cheia:	`int testaCheia(pPE p);`
retorna o tamanho (número de elementos) inseridos na pilha:	`int numElementos(pPE p);`

Tabela 1: Interface do TDA PE.

Bibliografia

[1] Backus, John. Evolução das Principais Linguagens de Programação. In: Sebesta, R. W. Conceitos de Linguagens de Programação. 4ª ed. Porto Alegre: Editora Bookman, 2000. p. 49-110.

[2] Barr, Michael. Programming embedded Systems in C and C++. First Edition. Sebastopol: O'Reilly and Associates, 1999. 174p. cap 1.: Introduction: C: The least Common Denominator.

[3] Fishwick, P. A. Computer Simulation Potentials, IEEE , Volume: 15 , Issue: 1 , P:24–27.Feb.-March.1996.

[4] Gersting, Judith L. "Fundamentos Matemáticos para a Ciência da computação". Ed. LTC.

[5] Goldberg, Adele. Suporte para Programação Orientada a Objeto. In: Sebesta, R. W. Conceitos de Linguagens de Programação. 4ª ed. Porto Alegre: Editora Bookman, 2000. p. 417-466.

[6] Horowitz, E. & Sahni, S. "Fundamentos de Estruturas de Dados". Ed. Campus. 1984.

[7] Pereira, Sílvio L. Estruturas de Dados Fundamentais – Conceitos e Aplicações. 7a. ed. Érica, 2003,p. 73

[8] Parnas, D. L. On the Criteria to be Used in Decomposing systems into Modules Communications of the ACM**,** Volume 15 Issue 12. December 1972.

[9] Ritchie, Dennis. Subprogramas. In: Sebesta, R. W. Conceitos de Linguagens de Programação. 4ª ed. Porto Alegre: Editora Bookman, 2000. p. 315-359.

[10] Szwarcfiter, Jayme L. & Markenzon L. “Estruturas de Dados e Seus Algoritmos”. Rio de Janeiro. Ed. LTC. 1994.

[11] Tenembaum, Aaron M. e outros. “Estruturas de Dados Usando C”. São Paulo. Ed.Makron Books. 1995.

[12] Tremblay, J.; Bunt, R..Ciência dos Computadores. Uma abordagem algorítmica, McGraw Hill, 1983.

[13] Van Gelder, Allen. A Discipline of Data Abstraction using ANSI C. Documento disponível em http://www.cse.ucsc.edu/~avg/. 02/2005.

[14] Veloso, Paulo et al. “Estruturas de Dados”. Ed. Campus. 1984.

[15] Wirth, Niklaus. “Algorithms + Data structures = Programs”, Ed. Prentice Hall, 1976.