SISTEMA EDUCACIONAL: Aplicação de Estrutura de Dados com Filas, Lista encadeada e Duplamente encadeada.

Kácio Felipe
Discente do Curso de Ciência da Computação
Instituto Federal de Brasília (IFB)

Caleul Raposo Caixeta
Discente do Curso de Ciência da Computação
Instituto Federal de Brasília (IFB)

RESUMO

Este projeto tem como objetivo realizar um levantamento bibliográfico referente às características e o contexto da linguagem de programação no uso de Listas Encadeadas Dinâmicas, Listas Duplamente Encadeadas Dinâmicas e Filas. è abordado o conceito de programação na linguagem C, a estrutura de programação no uso de Listas Encadeadas, Listas Duplamente Encadeadas e Filas. O artigo discute ainda o contexto de alocação de memória, ponteiro, uma síntese no uso de Listas Dinâmicas e aplicações que utilizam este recurso, tais como controle de empréstimos.

PALAVRAS-CHAVE: Listas Encadeadas, Listas Duplamente Encadeadas, Filas, Controle.

INTRODUÇÃO

A linguagem de programação C é uma evolução de duas linguagens anteriores, BCPL e B. BCPL foi criada para a escrita de sistemas operacionais e compiladores. Destaca no quesito de sistemas operativos as versões iniciais do kernel[1] do Unix[2] que foram desenvolvidas inicialmente em linguagem B. Como B e BCPL não tinham em sua estrutura de programação a definição dos tipos de dados ficará a cargo do programador realizar esta tratativa de seus tipos, conforme DEITEL & DEITEL (2001, p.56).

Em linha de sucessão, a linguagem C foi idealizada por Dennis Ritchie e originalmente implementada em um computador em 1972. A linguagem traz consigo a introdução dos tipos de dados, ausente nas linguagens B e BCPL. Ficou conhecida como a linguagem de programação usada para implementação das novas versões do Kernel[1] Unix[2] e de diversos núcleos de sistemas operativos da época, DEITEL & DEITEL (2001, p.56).

A linguagem aborda conceitos de programação importantes como Estruturas de Dados Struct[3] , Pilha[4] , Fila[5] e Lista[6] , alocações estáticas de memória, ou seja, reserva de memória pré-fixada, alocações dinâmicas de memória, onde o limite da memória para alocação pelas aplicações é a própria memória existente e disponível no computador, LAUREANO (2008, p.83).

A motivação da pesquisa sobre a linguagem C no uso de Listas Encadeadas Dinâmicas, Listas Duplamente Encadeadas e Filas foi devido à grande importância despendida pelo tema, como seu limite de usabilidade, os métodos de alocação utilizados, estes fixos ou dinâmicos, o uso de ponteiros, sendo este uma variável que guardará um endereço de memória, ou seja, o ponteiro aponta para uma posição de memória no computador e a implementação da linguagem no desenvolvimento de aplicações como controle de empréstimos, conforme LAUREANO (2008, p.11).

O objetivo que levou a produção deste artigo foi elucidar tópicos relevantes no uso de Listas Encadeadas Dinâmicas, Listas Duplamente Encadeadas e Filas. Desta forma a metodologia de pesquisa do artigo é um levantamento bibliográfico e está dividida em cinco seções. A primeira seção é destinada a definição da linguagem no uso de alocação estática versus alocação dinâmica, lista encadeada, lista duplamente encadeada e filas. A segunda seção será abordada a aplicabilidade desses tipos de lista e da fila.

As funcionalidades dessas listas e filas ficam destinadas a terceira seção. A quarta seção descreve um caso de abstração de dados o chamado struct e por último a quinta seção aborda um trecho de código com especificações de como implementar esta estrutura em linguagem de programação C, aborda ainda a iniciação de uma lista, os métodos de inserção, como exibir, imprimir na tela, os dados dessas listas e filas e execução de uma aplicação com uso desta estrutura.

---

[1]Kernel: é o núcleo do sistema operacional, responsável pela gerência dos recursos do hardware usados pelas aplicações, MAZIERO (2013, p.08).

[2]Unix: Sistema Operacional criado em 1969 por Ken Thompson e Dennis Ritchie, pesquisadores dos Bell Labs, MAZIERO (2013, P.25).

[3]Struct ou Estrutura de Dados: é uma estrutura ou registro, dependendo da linguagem de programação utilizada, designado por um grupo de itens, onde cada item tem um identificador próprio, sendo cada identificador, um membro, ou um campo da estrutura, LANGSAM, AUSGENSTEIN & TANENBAUM (1995, p.57).

[4]Pilha: é o inverso de uma fila porque usa o acesso último a entrar, primeiro a sair, o que as vezes é chamado de LIFO (Last In, First Out), SCHILDT (1996, P.535).

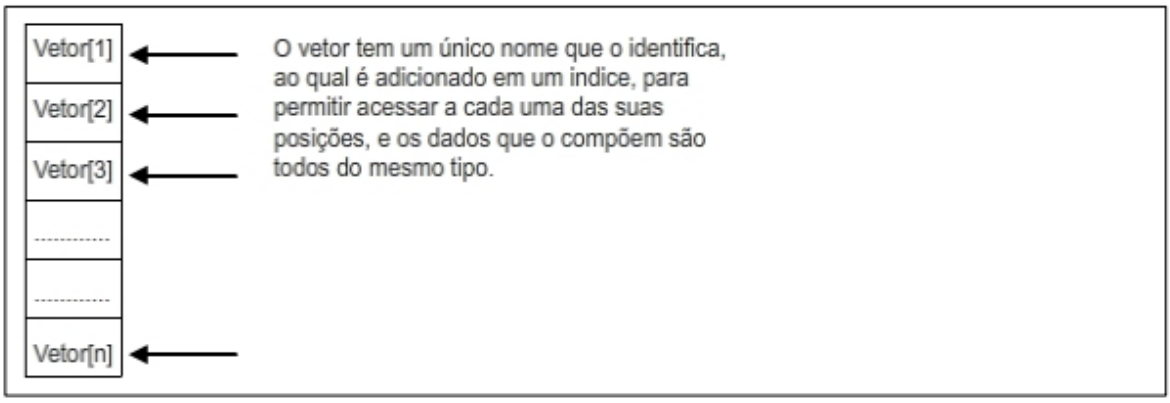
[5]Fila: é uma lista linear de informações, que é acessada na ordem primeiro a entra, primeiro a sair, SCHILDT (1996, P.526).

[6]Lista: é uma corrente que pode trabalhar com seu acesso de forma randômica, porque cada porção de informação carrega consigo um elo, ponteiro, ao próximo item de dados nesta corrente, SCHILDT (1996, P.540).

1. DEFINIÇÃO

De acordo com LAUREANO (2008, p.19), em uma alocação estática ao exemplo à declaração de um vetor, torna-se necessário dimensioná-lo, ou seja, prever a quantidade de espaço a ser ocupada pelo software durante a codificação. Tal definição é um fator limitante na dinâmica de espaços de memória do computador.

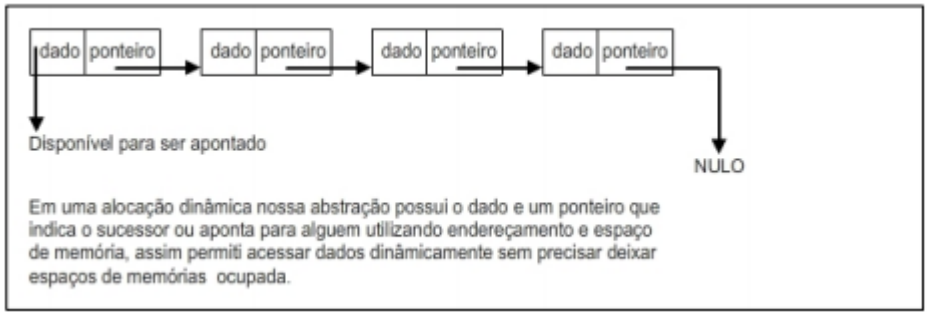
Figura 1: Exemplo ilustrado de uma alocação estática utilizando vetor.



Fonte: Adaptado de LAUREANO (2008, p.19).

A alocação dinâmica se assemelha a um vetor, porém sem desperdício de memória, isso se deve devido a requisição de memória ao sistema, em tempo de execução, um espaço de um determinado tamanho. Os dados se mantêm reservados até serem liberados pelo programa, disponibilizando novos espaços mantendo o tamanho devidamente ocupado não fixo, LAUREANO (2008, P19).

Figura 2: Exemplo ilustrado de uma alocação dinâmica utilizando lista simples.



Fonte: Adaptado de LAUREANO, (2008, p.19).

> “Uma lista é uma estrutura de dados dinâmica. O número de nós de uma lista pode variar consideravelmente à medida que são inseridos e removidos elementos. A natureza dinâmica de uma lista pode ser comparada à natureza estática de um vetor, cujo tamanho permanece constante”, TANEMBAUM (1995, p.225).

Desta forma, uma estrutura é uma lista, onde cada elemento aponta para o seu sucessor e o último elemento aponta para o primeiro da "Lista". Cada elemento é tratado como um ponteiro que é alocado dinamicamente a medida das inserções ou remoções dos dados, para isso utiliza-se "ponteiro para ponteiro", fica assim mais fácil o manuseio para mudança do dado inicial, TANEMBAUM (1995, p.280). A Estrutura é flexível e dinâmica.

> “Listas são estruturas muito flexíveis porque podem crescer ou diminuir de tamanho durante a execução de um programa, de acordo com a demanda. Itens podem ser acessados, inseridos ou retirados de uma lista. Listas são adequadas para aplicações onde não é possível prever a demanda por memória, permitindo a manipulação de quantidades imprevisíveis de dados, de formato também imprevisível”, ZIVIANI (1999, p.35).

O uso de Estrutura é vantajoso para economia de memória, vista que diferentemente de vetores estáticos, em aplicações o uso da lista se torna conveniente por não precisar de um tamanho ou previsão do crescimento da lista, além de em tempo de execução é alojado ou liberado espaço de memória para novos dados, ZIVIANI (1999, p.38).

Lista Duplamente Encadeada É um tipo de lista encadeada que pode ser vazia ou que pode ter um ou mais nós, sendo que cada nó possui dois ponteiros: um que aponta para o nó anterior e outro que aponta para o próximo nó. O importante é que, neste tipo de lista, o ponteiro externo pode apontar para qualquer nó da lista, pois é possível caminhar para a direita ou para a esquerda com igual facilidade. No Ponteiro Anterior. Ponteiro Próximo. Na estrutura de fila, os acessos aos elementos também seguem uma regra. O que diferencia a fila da pilha é a ordem de saída dos elementos: enquanto na pilha “o último que entra é o primeiro que sai”, na fila “o primeiro que entra é o primeiro que sai” (a sigla FIFO – first in, first out – é usada para descrever essa estratégia).

2. APLICAÇÃO EM SISTEMAS DIGITAIS

Um sistema digital é um conjunto de dispositivos de transmissão, processamento ou armazenamento de sinais digitais que usam valores discretos (descontínuos). Em contraste, os sistemas não-digitais (ou analógicos) usam um intervalo contínuo de valores para representarem informação.

Devido à natureza de listas encadeadas e duplamente encadeadas, essas estruturas de dados são bem eficazes quando se pensa em uma implementação de cadastro (independente de quantos tipos e quantos usuários) pelo cliente.

Consequentemente as filas tem sua determinada importância quanto à criação de uma lista de chamada por ordem de chegada, ou também uma lista de empréstimos por ordem de pedido.

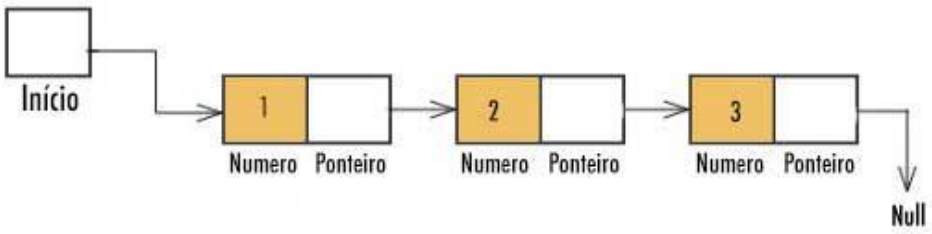
3. SITUAÇÃO PROBLEMA

Percebe-se hoje que todas as escolas, universidades e instituições de ensino antes de começarem a funcionar precisa ter uma estrutura muito bem organizada para receber tanto profissionais, como alunos para fazerem parte dela, já que todos os andamentos e princípios que envolvem a instituição necessita de uma integração entre ambos os públicos, dessa forma a maior dor que é percebida e enfrentada pelas instituições está em como ela realizará o controle de todas as ações de seus públicos, entendendo exatamente quem são seus alunos, seus professores e o que eles fazem na universidade, evitando ao máximo a utilização de papéis e instrumentos manuais já que os mesmos quando estão em uma quantidade exacerbada começam a ocupar muito espaço, se tornam algo desorganizado e ainda não segue as tendências de preservação mundial.

4. FUNCIONALIDADE

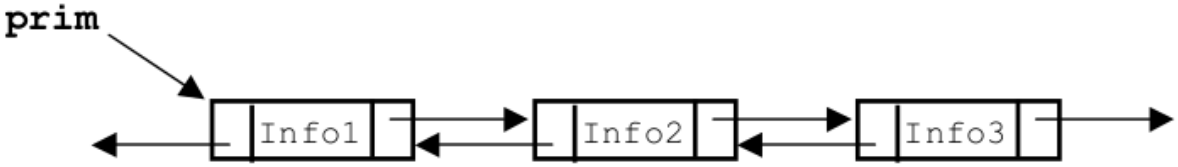
Numa lista encadeada, para cada novo elemento inserido na estrutura, alocamos um espaço de memória para armazená-lo. Desta forma, o espaço total de memória gasto pela estrutura é proporcional ao número de elementos nela armazenados.

Figura 3: Exemplo ilustrado de uma lista encadeada simples.



Em listas duplamente encadeadas cada elemento tem um ponteiro para o próximo elemento e um ponteiro para o elemento anterior. Desta forma, dado um elemento, podemos acessar ambos os elementos adjacentes: o próximo e o anterior. Se tivermos um ponteiro para o último elemento da lista, podemos percorrer a lista em ordem inversa, bastando acessar continuamente o elemento anterior, até alcançar o primeiro elemento da lista, que não tem elemento anterior (o ponteiro do elemento anterior vale NULL).

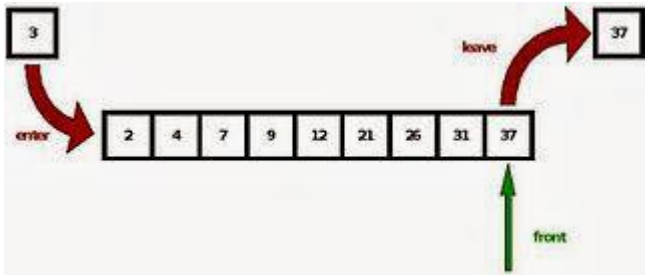
Figura 4: Exemplo ilustrado de uma lista duplamente encadeada.



Os primeiros elementos a chegar serão os primeiros a serem atendidos. Um termo muito conhecido para designar tal tipo de ideia é FIFO - First In, First Out (Primeiro que entra, primeiro que sai).

Em termos de programação, dizemos que os elementos que chegam vão para a cauda da fila, ou seja, para o final, serão atendidos por último. Os elementos que serão primeiro atendidos são os que estão na cabeça da fila (na frente).

Figura 5: exemplo de uma estrutura de dados do tipo fila.



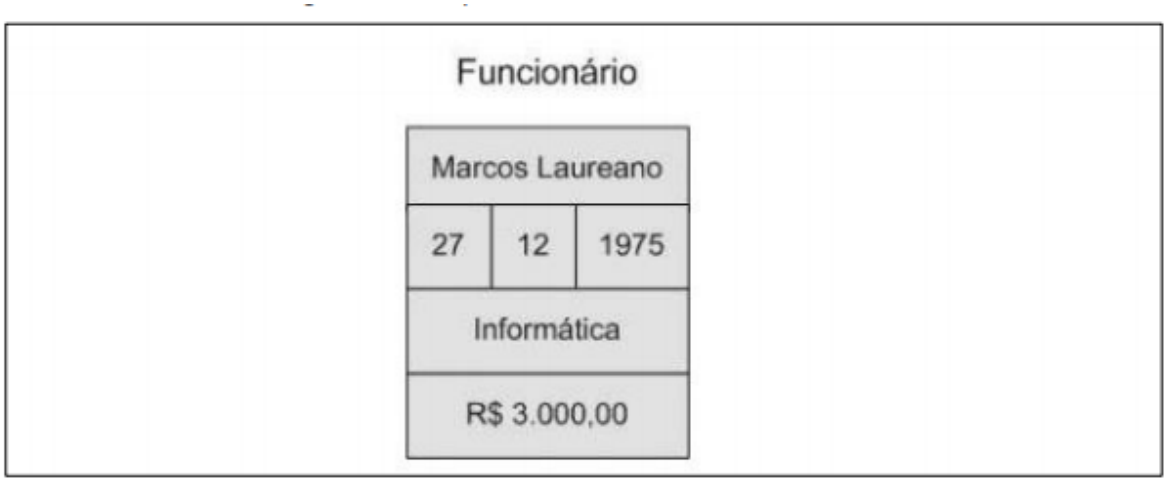
5. ABSTRAÇÃO

Conforme LAUREANO (2008, p.16), uma struct é uma estrutura que contém diversas outras variáveis e normalmente de tipos diferentes, as variáveis internas contidas são denominadas membros da struct. Podemos dizer que as structs da linguagem C são o equivalente ao que se denomina registros em outras linguagens de programação. Essas

estruturas podem ter quantos elementos você queira e dos tipos que você quiser definindo a estrutura e seus atributos, esses detalhes são definidos somente uma vez, na declaração da abstração.

O conceito de registro é muito utilizado quando temos elementos em nossos programas que precisam e fazem uso de vários tipos de variáveis e características. Utilizando, podemos trabalhar com vários tipos de informações de uma maneira mais fácil, rápida e organizada, uma vez que não temos que nos preocupar em declarar e decorar o nome de cada elemento contido, LAUREANO (2008, p.16).

Figura 6: Exemplo ilustrado de uma estrutura struct.



Fonte: LAUREANO (2008, p.17)

Figura 7: Exemplo de estrutura struct.

```
struct Funcionario
{
    char nome [40];
    struct
    {
        int dia;
        int mes;
        int ano;
    } dataNasc;
    char departamento[10];
    float salario;
};
```

Fonte: LAUREANO (2008, p.17)

6. DOCUMENTAÇÃO

Em uma lista encadeada e duplamente encadeada existem quatro execuções básicas para sua funcionalidade, são as funções de inicialização, inserção, remoção e consulta.

Para utilizar uma lista, primeiramente deve-se criar uma abstração da mesma, também conhecido como registro ou Struct. Dentro do struct deve-se conter o tipo de dado à inserção da lista e um ponteiro para auxiliar o apontamento para a próxima célula (item).

Figura 8: Exemplo de código de registro de uma lista encadeada.

```
struct lista {
   int info;
   struct lista* prox;
};

typedef struct lista Lista;
```

Figura 9: Exemplo de código de registro de uma lista duplamente encadeada.

```
struct lista2 {
   int info;
   struct lista2* ant;
   struct lista2* prox;
};

typedef struct Lista2 Lista2;
```

Figura 10: Exemplo de código de registro de uma fila.

```
#define N 100

struct fila {
   int ini, fim;
   float vet[N];
};
```

Na utilização de listas encadeadas e duplamente encadeadas devemos inicializá-las com apontamento NULL (nulo), pois ainda não há dados adicionados.

Figura 11: Exemplo de código de inicialização de uma lista encadeada ou duplamente encadeada.

```
/* função de inicialização: retorna uma lista vazia */
Lista* inicializa (void)
{
   return NULL;
}
```

A função para criar a fila aloca dinamicamente essa estrutura e inicializa a fila como sendo vazia, isto é, com os índices “ini” e “fim” iguais entre si (no caso, usamos o valor zero).

Figura 12: Exemplo de código de inicialização de uma fila.

```
Fila* cria (void)
{
   Fila* f = (Fila*) malloc(sizeof(Fila));
   f->ini = f->fim = 0;   /* inicializa fila vazia */
   return f;
}
```

Em função de inserção à lista encadeada e duplamente encadeada, cria-se uma nova lista de auxílio, ela é de extrema importância pois assim não se é manipulado dados da lista original, a partir da auxiliar é feito a troca de apontamento, fazendo com que a nova lista com os novos dados aponte para a antiga lista, e no caso da duplamente encadeada, a antiga também aponta para a nova.

Figura 13: Exemplo de código de inserção de uma lista encadeada

```
/* inserção no início: retorna a lista atualizada */
Lista* insere (Lista* l, int i)
{
   Lista* novo = (Lista*) malloc(sizeof(Lista));
   novo->info = i;
   novo->prox = l;
   return novo;
}
```

figura 14: exemplo de código de inserção de uma lista duplamente encadeada.

```c
/* inserção no inicio */
Lista2* insere (Lista2* l, int v)
{
   Lista2* novo = (Lista2*) malloc(sizeof(Lista2));
   novo->info = v;
   novo->prox = l;
   novo->ant = NULL;
   /* verifica se lista não está vazia */
   if (l != NULL)
      l->ant = novo;
   return novo;
}
```

Para inserir um elemento na fila, usamos a próxima posição livre do vetor, indicada por “fim”. Devemos ainda assegurar que há espaço para a inserção do novo elemento, tendo em vista que trata-se de um vetor com capacidade limitada. Consideraremos que a função auxiliar que faz o incremento circular está disponível.

Figura 15: Exemplo de código de inserção de uma fila.

```c
void insere (Fila* f, float v)
{
   if (incr(f->fim) == f->ini) {    /* fila cheia: capacidade esgotada
*/
      printf("Capacidade da fila estourou.\n");
      exit(1);          /* aborta programa */
   }
   /* insere elemento na próxima posição livre */
   f->vet[f->fim] = v;
   f->fim = incr(f->fim);
}
```

Uma possível implementação da função para retirar um elemento da lista é mostrada a seguir. Inicialmente, busca-se o elemento que se deseja retirar, guardando uma referência para o elemento anterior.

Figura 16: Exemplo de código de remoção de um elemento de uma lista encadeada.

```c
/* função retira: retira elemento da lista */
Lista* retira (Lista* l, int v) {
   Lista* ant = NULL; /* ponteiro para elemento anterior */
   Lista* p = l;      /* ponteiro para percorrer a lista*/

   /* procura elemento na lista, guardando anterior */
   while (p != NULL && p->info != v) {
      ant = p;
      p = p->prox;
   }

   /* verifica se achou elemento */
   if (p == NULL)
      return l;   /* não achou: retorna lista original */

   /* retira elemento */
   if (ant == NULL) {
      /* retira elemento do inicio */
      l = p->prox;
   }
   else {
      /* retira elemento do meio da lista */
      ant->prox = p->prox;
   }
   free(p);
   return l;
}
```

A função de remoção de lista duplamente encadeada é mais complicada, pois temos que acertar o encadeamento duplo. Em contrapartida, podemos retirar um elemento da lista conhecendo apenas o ponteiro para esse elemento. Desta forma, podemos usar a função de busca acima para localizar o elemento e em seguida acertar o encadeamento, liberando o elemento ao final.

Se p representa o ponteiro do elemento que desejamos retirar, para acertar o encadeamento devemos conceitualmente fazer:

```
p->ant->prox = p->prox;
p->prox->ant = p->ant;
```

A função para retirar o elemento do início da fila fornece o valor do elemento retirado como retorno. Podemos também verificar se a fila está ou não vazia.

Figura 17: Exemplo de código de remoção do início da fila.

```
float retira (Fila* f)
{
   float v;
   if (vazia(f)) {
      printf("Fila vazia.\n");
      exit(1);         /* aborta programa */
   }
   /* retira elemento do início */
   v = f->vet[f->ini];
   f->ini = incr(f->ini);
   return v;
}
```

Para consulta de algum dado específico dentro de uma lista encadeada ou duplamente encadeada, precisamos de alguma informação compatível para encontrar tal dado com especificidade, buscando com uma auxiliar dentro da lista enquanto não encontra, sempre pulando para o próximo endereço definido na própria lista.

Figura 18: Exemplo de código de consulta de algum elemento dentro de uma lista.

```
void consultarAluno(Aluno* a){
    Aluno* aux;
    double cpf;

    printf("\nDigite o cpf do aluno para consulta:\n");
    scanf("%lf", &cpf);

    for(aux = a; aux != NULL; aux = aux->prox){
        if(aux->CpF == cpf){
            printf("\nDados:\nNome: %s\n", aux->Nome);
            printf("Data de nascimento: %lf\n", aux->DataNascimento);
            printf("CPF: %lf\n", aux->CpF);
            return;
        };
    };
    printf("Aluno não cadastrado\n");
};
```

CONSIDERAÇÕES FINAIS

Em razão do contexto abordado sobre Listas encadeadas, duplamente encadeadas e filas, foi possível elucidar quais são as características relevantes no uso destas estrutura aplicado no desenvolvimento de sistemas digitais e a dinâmica de alocação de espaço de memória e acesso a estes dados na memória do computador, desta forma é possível utilizar estas estruturas dinâmicas ao invés de estruturas estáticas devido a sua natureza de performance.

A lista encadeada dinâmica, duplamente encadeada e filas são perfeitas para a aplicação de sistemas digitais, pois são capazes de cadastrar o que deve ser cadastrado, fazer filas utilizando a estrutura padrão deste e preenchendo-as com dados já adicionados.

REFERÊNCIA BIBLIOGRÁFICA

Szwarcfiter, J. L.; Markenzon, L. ESTRUTURA DE DADOS E SEUS ALGORITMOS. 3ª ed. Rio de Janeiro: LTC, 2010. 318 p.

Celes, Waldemar et al. INTRODUÇÃO A ESTRUTURA DE DADOS: COM TÉCNICAS DE PROGRAMAÇÃO EM C. 2ª ed. Rio de Janeiro: Elsevier, 2004. 410 p.

LAUREANO, Marcos. Estruturas de Dados com Algoritmos em C. Rio de Janeiro: Brasport, 2008.

SAVI, Rafael. Avaliação de Jogos Voltados Para a Disseminação do Conhecimento. In: Tese (doutorado) - Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Engenharia e Gestão do Conhecimento. Florianópolis, 2011.

TANEMBAUM, Aaron M; LANGSAM, Yedidyah Langsam; AUGENSTEIN, Moshe J. Augenstein. Estruturas de Dados Usando C. São Paulo: MARKON Books, 1995.

ZIVIANI, Nivio. Projeto de Algoritmos com Implementações em Pascal e C. São Paulo: Editora Pioneira, 1999.