

FELIPE HIROSHI CORREIA KATSUMOTO

**ALGORITMOS E ESTRUTURAS DE DADOS**

Controle de Sala de Aula

Sumário

**Nenhuma entrada de sumário foi encontrada.**

# Estruturas básicas

## Aluno

1. struct nota;
2. typedef struct aluno
3. {
4. int matricula;
5. int anoEntrada;
6. int numFaltas;
7. float somaNotas;
8. char nome[50];
9. char curso[50];
10. struct nota \*notas;
11. } Aluno;

## Notas

1. typedef struct nota {
2. int id;
3. float total;
4. float chave;
5. struct nota \*prox;
6. } Nota;

## Tabela Hash

1. typedef struct lista
2. {
3. Aluno \*aluno;
4. int prox;
5. } Lista;
6. typedef Lista \*Hash;

# Função Hash

1. int hash(int valor){
2. return valor%100;
3. }

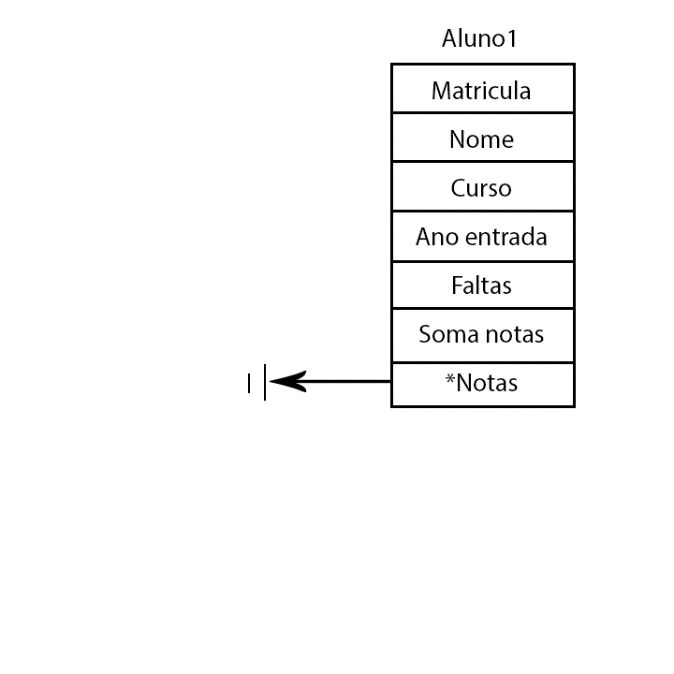
Considerando que o valor para a hash será o da matrícula e que a função deverá se adequar ao modelo de geração de matricula, considerei um modelo simples que funciona da maneira AAPSS, sendo

* AA – Ano de entrada na instituição, exemplo 23, para 2023. 22, para 2022;
* P – Período de entrada na instituição, 1 ou 2, em caso de divisão semestral;
* SS – É um número Sequencial que representa o número posterior ao último aluno incluído na matrícula do curso.

E, como está se tratando de uma sala de aula apenas, a grande maioria dos alunos terão ano de entrada e período idênticos, portanto, o SS será pouco repetido, com isso sendo muitas vezes o próprio índice.

# Alunos e Notas

A representação da struct Aluno, é a seguinte:



Portanto, fica definido que o campo Notas será uma lista encadeada, uma vez que o enunciado define que “cada aluno tem um conjunto de notas (de tamanho variável)”.

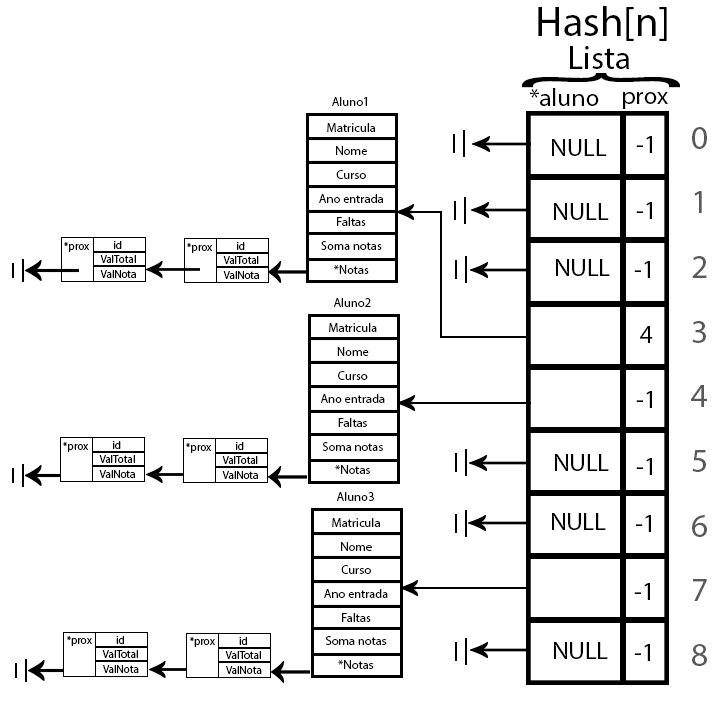
# Tratamento de colisões na tabela hash

# Encadeamento Interior (listas coligadas)

Foi escolhido o tipo de tratamento de colisões por listas coligadas devido a necessidade de ordenação da tabela para geração de relatórios baseados em comparações de variáveis arbitrariamente escolhidas pelo usuário.

É esperado que a tabela hash seja representada da seguinte maneira após algumas poucas inserções:

Imagem representação tabela hash, alunos, notas



## Funções relacionadas a Hash

### Inserir

1. void inserir(Hash \*tabela, Aluno \*aluno){
2. int indice = hash(aluno->matricula);
3. if(tabela[indice]->aluno!=NULL){
4. printf("entrou??");
5. int i;
6. for (i = (indice+1); tabela[i]->aluno!=NULL && i != indice; i = (i + 1) % 100);
7. if (i == indice) {
8. printf("Tabela hash cheia, não foi possível inserir %s\n", aluno->nome);
9. return;
10. }
11. while(tabela[indice]->prox!=-1){
12. indice = tabela[indice]->prox;
13. }
14. tabela[indice]->prox = i;
15. tabela[i]->aluno = aluno;
16. }
17. else {
18. tabela[indice]->aluno = aluno;
19. }
20. }

A função de inserção busca um índice válido de maneira sequencial até encontrar um espaço vazio ou até que o índice atual seja igual ao atual, isto significa que a tabela hash está cheia.

Uma vez encontrado o próximo índice livre i, percorremos a lista coligada até o final usando o campo prox, que aponta para o próximo elemento colidido. Quando o último índice é encontrado, o campo prox do último elemento é atualizado para apontar para o novo índice i, onde o aluno será inserido.

### Busca

1. Aluno \*busca(int chave, Hash \*tabela){
2. int indice = hash(chave);
3. while(indice!=-1){
4. if(tabela[indice]->aluno->matricula == chave) return tabela[indice]->aluno;
5. indice = tabela[indice]->prox;
6. }
7. return NULL;
8. }

A função busca recebe o valor a ser procurado, neste caso, para a utilização correta o número da matricula de algum aluno. Com a chave a ser procurada, esta vai para a função de hashing para obter o índice inicial, a partir deste, a função percorre os encadeamentos interiores disponíveis até encontrar o aluno cuja matricula foi a passada inicialmente. Caso não ache, retorna NULL.

### Imprime

1. void imprime(Hash \*tabela) {
2. for(int i=0; i<100; i++){
3. if(tabela[i]->aluno==NULL) continue;
4. printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");
5. printf("Matricula: %d\n", tabela[i]->aluno->matricula);
6. printf("Nome: %s\n", tabela[i]->aluno->nome);
7. printf("Soma de Notas: %.2f\n", tabela[i]->aluno->somaNotas);
8. printf("Numero de faltas: %d\n", tabela[i]->aluno->numFaltas);
9. printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");
10. }
11. }

A função de imprime será utilizada principalmente posteriormente na geração de relatório organizado por algum campo da struct aluno. A função percorre a tabela hash de maneira sequencial e vai imprimindo, para evitar erros de segmentação ou de imprimir vazios, caso encontre uma posição vazia, ele vai para a próxima iteração.

# Chamada

O enunciado define que há um número máximo de 18 aulas a serem cadastradas e, caso o número de faltas seja superior a 10, o programa deverá imprimir uma mensagem informando que o aluno repetiu.

A função de chamada ficou da seguinte maneira:

1. void chamada(Hash \*tabela){
2. printf("Chamada\nPara aluno presente, aperte P\nPara aluno ausente, aperte F\n");
3. for(int i=0; i<100; i++){
4. if(tabela[i]->aluno==NULL) continue;
5. char resp = '\n';
6. printf("%s: ", tabela[i]->aluno->nome);
7. while (resp == '\n') resp = getchar();
8. if(resp=='F'||resp=='f') tabela[i]->aluno->numFaltas++;
9. if (tabela[i]->aluno->numFaltas >= 10) printf("Aluno reprovado por faltas!");
10. }
11. }

Sendo a limitação de 18 aulas levada em consideração no main.c da seguinte maneira:

1. if(numChamada>18) printf("Numero de aulas maximas cadastradas");
2. else{
3. chamada(tabela);
4. numChamada++;
5. }

O enunciado ainda define que, caso haja chamadas cadastradas e seja necessário cadastrar um aluno novo, faça as chamadas já registradas para o aluno.

Portanto, a função que recupera as chamadas para o novo aluno é:

1. void recuperaChamada(Aluno \*aluno, int num){
2. for(int i=0; i<num; i++){
3. char resp = '\n';
4. printf("Chamada %d do %s: ", i+1, aluno->nome);
5. while (resp == '\n') resp = getchar();
6. if (resp == 'F' || resp=='f') aluno->numFaltas++;
7. if (aluno->numFaltas >= 10) printf("Aluno reprovado por faltas!");
8. }
9. }

Conferindo novamente no main por meio do controlador numChamadas quantas chamadas já foram registradas.

# Notas e avaliações

O campo notas dentro da struct aluno é uma lista encadeada simples com uma inserção sempre no final, considerando o cenário apresentado que tem um máximo de 18 aulas, a escolha de notas não ter um tail foi considerada pois não atingirá um número de avaliações considerável para causar uma perda computacional, ainda que a inserção tenha ido para O(n).

## Avaliações

As avaliações são controladas de maneira que cada avaliação recebe um id auto incremental que diferenciará as avaliações para, posteriormente, tenha um controle de qual avaliação será pedido o relatório de notas.

1. void criaAvaliacao(Hash \*tabela, int \*id)
2. {
3. float notaMax, notaVar = -1;
4. (\*id)++;
5. printf("Quanto vale a avaliacao? ");
6. scanf("%f", &notaMax);
7. for (int i = 0; i < 100; i++)
8. {
9. if(tabela[i]->aluno==NULL) continue;
10. while (notaVar < 0 || notaVar > notaMax) {
11. printf("Digite a nota do aluno %s ", tabela[i]->aluno->nome);
12. scanf("%f", &notaVar);
13. }
14. insereNota(notaVar, notaMax, tabela[i]->aluno, \*id);
15. notaVar = -1;
16. }
17. }

Assim como a chamada, caso algum aluno novo seja adicionado após a existência de avaliações já cadastradas, o aluno deverá conter as avaliações, desta maneira a seguinte função se faz necessária.

1. void recuperaAvaliacao(int num, Aluno \*aluno, Hash\* tabela){
2. float nota, valTotal;
3. Nota \*aux;
4. //pega um aluno para usar de referência para valTotal
5. for(int i=0; i<100; i++){
6. if(tabela[i]->aluno==NULL || tabela[i]->aluno->notas==NULL) continue;
7. aux = tabela[i]->aluno->notas;
8. break;
9. }
10. for(int i=1; i<=num; i++){
11. valTotal = aux->total;
12. printf("Digite a nota da avaliacao %d: ", i);
13. scanf("%f", &nota);
14. insereNota(nota, valTotal, aluno, i);
15. aux = aux->prox;
16. }
17. }

# Algoritmos de ordenação

## Selection Sort (O(n²))

O algoritmo de ordenação O(n²) escolhido foi o selection sort porque sua implementação é simples.

1. void troca(Hash \*tabela, int a, int b)
2. {
3. if (tabela[a]->aluno != NULL && tabela[b]->aluno != NULL) {
4. Aluno\* aux = tabela[a]->aluno;
5. tabela[a]->aluno = tabela[b]->aluno;
6. tabela[b]->aluno = aux;
7. }
8. }
9. void selectionSort(Hash \*tabela, int metodo) {
10. int menor;
11. for (int i = 0; i < 100; i++) {
12. if (tabela[i]->aluno == NULL) {
13. continue;
14. }
15. menor = i;
16. for (int j = i + 1; j < 100; j++) {
17. if (tabela[j]->aluno == NULL) {
18. continue;
19. }
20. if (metodo == 2) {  // Ordenar por nome
21. if (strcmp(tabela[j]->aluno->nome, tabela[menor]->aluno->nome) < 0) {
22. menor = j;
23. }
24. } else {  // Ordenar por matrícula
25. if (tabela[j]->aluno->matricula < tabela[menor]->aluno->matricula) {
26. menor = j;
27. }
28. }
29. }
30. if (menor != i) {
31. troca(tabela, i, menor);
32. }
33. }
34. }

## Selection Sort para listas encadeadas

Como será necessária uma impressão de notas de maneira inversamente ordenada, foi necessária uma adaptação para que ordene do maior para o menor e para que funcione em listas encadeadas.

Além disso, para que o programa extraia apenas as notas da avaliação desejada, faz-se necessário a função extraiNotas.

Nota \*extraiNotas(Hash \*tabela, int id)

{

    Nota \*inicio = NULL;

    Nota \*fim = NULL;

    for (int i = 0; i < 100; i++)

    {

        if (tabela[i]->aluno == NULL) continue;

        Aluno \*alunoAux = tabela[i]->aluno;

        Nota \*aux = alunoAux->notas;

        while (aux != NULL)

        {

            if (aux->id == id)

            {

                Nota \*novaNota = criaNota(aux->chave, aux->total, aux->id);

                if (inicio == NULL)

                {

                    inicio = novaNota;

                    fim = novaNota;

                }

                else

                {

                    fim->prox = novaNota;

                    fim = novaNota;

                }

            }

            aux = aux->prox;

        }

    }

    if (inicio == NULL)

    {

        printf("Nenhuma nota encontrada com o id fornecido.\n");

    }

    return inicio;

}

void trocaNota(Nota \*a, Nota \*b, Nota \*\*lista)

{

    Nota \*inicio = \*lista;

    if (a->chave == b->chave)

        return;

    Nota \*antA = NULL, \*A = inicio;

    Nota \*antB = NULL, \*B = inicio;

    while (A != NULL && A != a)

    {

        antA = A;

        A = A->prox;

    }

    while (B != NULL && B != b)

    {

        antB = B;

        B = B->prox;

    }

    if (antA != NULL)

        antA->prox = b;

    else

        \*lista = b;

    if (antB != NULL)

        antB->prox = a;

    else

        \*lista = a;

    Nota \*proxA = a->prox;

    a->prox = b->prox;

    b->prox = proxA;

}

void selectionNota(Nota \*\*lista)

{

    Nota \*maior;

    Nota \*aux, \*i = \*lista;

    if (i==NULL || i->prox ==NULL) return;

    while (i != NULL)

    {

        maior = i;

        aux = i;

        while (aux != NULL)

        {

            if (aux->chave > maior->chave)

            {

                maior = aux;

            }

            aux = aux->prox;

        }

        trocaNota(i, maior, lista);

        i = maior;

        i = i->prox;

    }

}

## Quick Sort (n log(n))

1. int partition(Hash \*tabela, int inicio, int fim, int metodo) {
2. Aluno\* pivot = tabela[fim]->aluno;
3. int i = inicio - 1;
4. for (int j = inicio; j < fim; j++) {
5. if (tabela[j]->aluno != NULL && pivot != NULL) {
6. int compara = 0;
7. if (metodo == 1) {  // Ordenar por soma de notas
8. compara = tabela[j]->aluno->somaNotas <= pivot->somaNotas;
9. } else if (metodo == 2) {  // Ordenar por nome
10. compara = strcmp(tabela[j]->aluno->nome, pivot->nome) <= 0;
11. } else {  // Ordenar por matrícula (padrão)
12. compara = tabela[j]->aluno->matricula <= pivot->matricula;
13. }
14. if (compara) {
15. i++;
16. Aluno\* temp = tabela[i]->aluno;
17. tabela[i]->aluno = tabela[j]->aluno;
18. tabela[j]->aluno = temp;
19. }
20. }
21. }
22. Aluno\* temp = tabela[i + 1]->aluno;
23. tabela[i + 1]->aluno = tabela[fim]->aluno;
24. tabela[fim]->aluno = temp;
25. return i + 1;
26. }
27. void quickSort(Hash \*tabela, int inicio, int fim, int metodo) {
28. if (inicio < fim) {
29. int pivotIndex = partition(tabela, inicio, fim, metodo);
30. quickSort(tabela, inicio, pivotIndex - 1, metodo);
31. quickSort(tabela, pivotIndex + 1, fim, metodo);
32. }
33. }

# Relatórios

## Relatório alunos

1. void relatorioAlunos(Hash \*tabela)
2. {
3. int ordem;
4. //faz uma copia para gerar o relatório
5. Hash \*copia = copiarLista(tabela);
6. imprime(copia);
7. printf("Ordenar por:\n1: Matricula\n2: Nome\n3: Soma de Notas\n4: Num. faltas:\n");
8. scanf("%d", &ordem);
9. switch (ordem)
10. {
11. case 1:
12. // organiza por matricula
13. selectionSort(copia, 1);
14. break;
15. case 2:
16. // organiza por nome
17. selectionSort(copia, 2);
18. break;
19. case 3:
20. // organiza por soma
21. quickSort(copia, 0, 99, 1);
22. break;
23. case 4:
24. // organiza por numero de faltas
25. quickSort(copia, 0, 99, 2);
26. break;
27. default:
28. // organiza por matricula
29. selectionSort(copia, 1);
30. break;
31. }
32. // matricula / nome / soma de notas / numero de faltas
33. imprime(copia);
34. }

Com as funções anteriores já implementadas, para gerar relatórios, basta organizar suas chamadas e, claro, adequar às vontades do usuário.

## Relatório notas

1. void relatorioNotas(Hash \*tabela, int numAvaliacoes)
2. {
3. if (numAvaliacoes == 0)
4. {
5. printf("Sem avaliacoes cadastradas");
6. return;
7. }
8. int id;
9. printf("Qual avaliacao voce quer o relatorio? Tem %d avaliacoes\n", numAvaliacoes);
10. scanf("%d", &id);
11. Nota \*notasRel = extraiNotas(tabela, id);
12. selectionNota(&notasRel);
13. Nota \*aux = notasRel;
14. if (aux != NULL)
15. {
16. // nota maxima
17. printf("Nota maxima: %.2f\n", aux->chave);
18. // nota média
19. printf("Nota media da avaliacao %d: %.2f\n", id, media(notasRel));
20. // percorre até o final para encontrar a nota mínima
21. while (aux->prox != NULL)
22. {
23. aux = aux->prox;
24. }
25. printf("Menor nota: %.2f\n", aux->chave);
26. // todas as notas
27. printf("\nTodas as notas:\n");
28. while (notasRel != NULL)
29. {
30. printf("%.2f\n", notasRel->chave);
31. notasRel = notasRel->prox;
32. }
33. }
34. else
35. {
36. printf("Nenhuma nota encontrada.\n");
37. }
38. }

Assim como no relatório alunos, a implementação do relatório de notas se torna banal, basta extrair uma lista encadeada com as notas da avaliação desejada e após isso, realizar uma impressão simples de lista encadeada.

# Arquivos









