UNI HELPER

Propósito da aplicação

Administração de informações acadêmicas

- adição/remoção de dados de alunos e docentes
- consulta de registros acadêmicos
- gestão de grades horárias e disciplinas

Opções

Administração de informações acadêmicas

- adição/remoção de dados de alunos e docentes
- consulta de registros acadêmicos
- gestão de grades horárias e disciplinas

Tipos

- uso de união discriminada
- tamanho fixo por registro

```c
typedef struct {                    // Tipo = Aluno
        long int matr;              // Matrícula
        char peri;                  // Período
        char discs;                 // Quantidade de disciplinas
        int grade[DIAS][TURNOS];// Grade de horários [i][j];
} Dados_aluno;

typedef struct {                    // Tipo = Professor
        int id;                     // Número de identificação do docente
        float slro;                 // Salário
        char turms;                 // Quantidade de turmas
        int aulas[DIAS][TURNOS];// Grade das aulas por dia e horário;
} Dados_professor;

typedef struct {
        short int uid;  // ID Único / UID (gerado automaticamente)
        char valido;    // Indicador de remoção
        // ~tipo~ do dado [Alun == Aluno / Prof == Professor]
        enum __attribute__ ((__packed__)) {Alun, Prof} tipo;
        char nome[TAM_NOME];

        union {
                Dados_aluno aluno;
                Dados_professor professor;
        } dados;
} Membro;
```

Armazenamento

```c
int Armazena(Membro * pss, FILE * f) {  // escreve os dados no arquivo
        size_t size_final = 0;
        size_final += sizeof(pss->uid);
        size_final += sizeof(pss->valido);
        size_final += sizeof(pss->tipo);
        size_final += sizeof(pss->nome);

        size_t size_aluno = 0;
        size_aluno += sizeof(pss->dados.aluno.matr);
        size_aluno += sizeof(pss->dados.aluno.peri);
        size_aluno += sizeof(pss->dados.aluno.discs);
        size_aluno += sizeof(pss->dados.aluno.grade);

        size_t size_professor = 0;
        size_professor += sizeof(pss->dados.professor.id);
        size_professor += sizeof(pss->dados.professor.slro);
        size_professor += sizeof(pss->dados.professor.turms);
        size_professor += sizeof(pss->dados.professor.aulas);

        if (size_aluno>size_professor)
                size_final += size_aluno;
        else
                size_final += size_professor;

        size_t byt = 0; // contagem de bytes escritos

        byt += writerb(&pss->uid, sizeof(pss->uid), 1, f);
        byt += writerb(&pss->valido, sizeof(pss->valido), 1, f);
        byt += writerb(&pss->tipo, sizeof(pss->tipo), 1, f);
        byt += writerb(pss->nome, sizeof(pss->nome), 1, f);
```

Armazenamento

```
    if (pss->tipo == Alun) {
        byt += writerb(&pss->dados.aluno.matr, sizeof(pss->dados.aluno.matr), 1, f);
        byt += writerb(&pss->dados.aluno.peri, sizeof(pss->dados.aluno.peri), 1, f);
        byt += writerb(&pss->dados.aluno.discs, sizeof(pss->dados.aluno.discs), 1, f);
        byt += writerb(pss->dados.aluno.grade, sizeof(pss->dados.aluno.grade), 1, f);
    }
    else {
        byt += writerb(&pss->dados.professor.id, sizeof(pss->dados.professor.id), 1, f);
        byt += writerb(&pss->dados.professor.slro, sizeof(pss->dados.professor.slro), 1, f);
        byt += writerb(&pss->dados.professor.turms, sizeof(pss->dados.professor.turms), 1, f);
        byt += writerb(pss->dados.professor.aulas, sizeof(pss->dados.professor.aulas), 1, f);
    }


    if (size_final > byt)
    {
        size_t fill = size_final - byt;
        for (size_t i = 0; i < fill; i++)
        {
            fputc(0, f);
        }
    }

    return (int) byt;
}
```

Leitura

```
Membro Le(long posicao_registro)
{
    Membro pss;

    FILE * f = fopen(ARQUIVO_DADOS, "rb");
    fseek(f, posicao_registro, SEEK_SET);

    fread(&pss.uid, sizeof(pss.uid), 1, f);
    fread(&pss.valido, sizeof(pss.valido), 1, f);
    fread(&pss.tipo, sizeof(pss.tipo), 1, f);
    fread(pss.nome, sizeof(pss.nome), 1, f);

    if (pss.tipo == Alun) {
        fread(&pss.dados.aluno.matr, sizeof(pss.dados.aluno.matr), 1, f);
        fread(&pss.dados.aluno.peri, sizeof(pss.dados.aluno.peri), 1, f);
        fread(&pss.dados.aluno.discs, sizeof(pss.dados.aluno.discs), 1, f);
        fread(pss.dados.aluno.grade, sizeof(pss.dados.aluno.grade), 1, f);
    }
    else {
        fread(&pss.dados.professor.id, sizeof(pss.dados.professor.id), 1, f);
        fread(&pss.dados.professor.slro, sizeof(pss.dados.professor.slro), 1, f);
        fread(&pss.dados.professor.turms, sizeof(pss.dados.professor.turms), 1, f);
        fread(pss.dados.professor.aulas, sizeof(pss.dados.professor.aulas), 1, f);
    }

    fclose(f);
    return pss;
}
```

Edição

```
long BuscarRegistroId()
{
        FILE * file_pointer = fopen(ARQUIVO_DADOS, "rb");

        char input[TAM_INPUT] = {0};
        short int id_unico;

        printf("\n Selecione o id único (UID): ");
        strin(input, TAM_INPUT, stdin);
        if (sscanf(input, "%hd", &id_unico) != 1 || id_unico < 0) {
                return -2;
        }

        long posicao_registro = id_unico * TAM_REGISTRO;

        fseek(file_pointer, 0L, SEEK_END);
        long end_file_pos = ftell(file_pointer);
        //printf(" end_file_pos: %lu\n", end_file_pos);

        if (posicao_registro >= end_file_pos) {
                fprintf(stderr, " Registro com o id único %hd inexistente\n",
                id_unico);
                fclose(file_pointer);
                return -1;
        }

        fclose(file_pointer);
        return posicao_registro;
}
```

Remoção

```c
void RemoverRegistro()
{
        // usuário fornece o UID à função, a função
        // retorna a posição de início do registro associado
        long posicao_registro = BuscarRegistroId();
        FILE * file_pointer = fopen(ARQUIVO_DADOS, "rb+");

        // armazena o UID antes de remover
        short uid;
        fseek(file_pointer, posicao_registro, SEEK_SET);
        fread(&uid, sizeof(uid), 1, file_pointer);

        char zeros[TAM_REGISTRO] = {0};

        // remove o registro (preenche com zeros)
        fseek(file_pointer, posicao_registro, SEEK_SET);
        fwrite(zeros, sizeof(char), TAM_REGISTRO, file_pointer);

        // restaura apenas o UID do registro
        fseek(file_pointer, posicao_registro, SEEK_SET);
        fwrite(&uid, sizeof(uid), 1, file_pointer);

        fclose(file_pointer);
}
```