

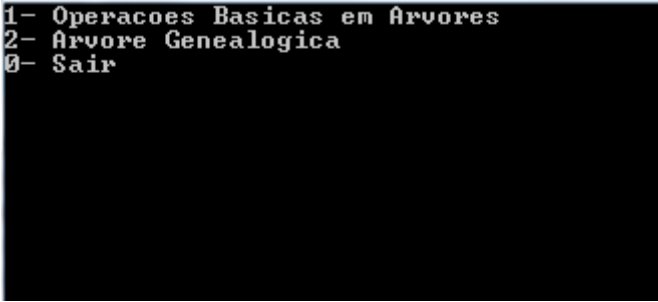
```
# SSC0300 - LINGUAGEM DE PROGRAMAÇÃO E APLICAÇÕES
# TRABALHO 2 - APLICAÇÕES DE ALGORITMOS DE ÁRVORES
```

```
# PARTICIPANTES DO GRUPO
- FABIANO CAVALHEIRO PAULELLI - 8670013
- RENAN STRANO DE OLIVEIRA - 8530980
```

```
# AMBIENTE DE DESENVOLVIMENTO
- PLATAFORMA: LINUX - Lubuntu - Baseado no Ubuntu
- COMPILADOR: GCC/CPP ( CODEBLOCKS )
```

```
# INSTRUÇÕES DE EXECUÇÃO DO PROGRAMA trab.c
```

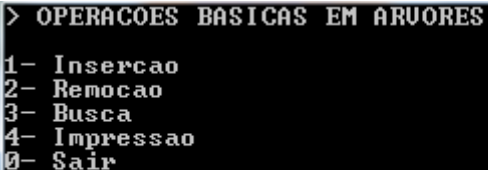
Ao executar o programa "trab.c", um menu irá abrir .



```
1- Operacoes Basicas em Arvores
2- Arvore Genealogica
0- Sair
```

>> PARTE 1:

A opção "1" ( Operações Basicas em Arvores ), irá levar para um menu secundário, que trará as opções de: Inserção, Remoção, Busca, Impressão e Sair.



```
> OPERACOES BASICAS EM ARVORES
1- Insercao
2- Remocao
3- Busca
4- Impressao
0- Sair
```

O usuário então, deverá, necessariamente, escolher a opção " 1 " do menu secundário para conseguir trabalhar de forma adequada. Caso contrário, se o usuário escolher qualquer outra opção senão a " 1 ", primeiramente, não conseguirá usar o programa de forma adequada, e uma mensagem de elemento inexistente aparecerá.

```
>> REMOCAO
Valor a ser Removido: 2
Elemento nao existe!

ENTER para continuar...
```

Ao inserir todos os elementos, o usuário, então, poderá trabalhar de forma natural no programa. Exemplo:

( INSERINDO OS ELEMENTOS ):

Primeiramente, o usuário deverá escolher a quantidade de elementos a ser adicionado e logo após, inserir valor por valor. Se os elementos forem inseridos, uma mensagem de sucesso aparecerá na tela. Caso o elemento exista, uma mensagem advertará o usuário.

```
>> INSERCAO
Quantidade de Elementos: 4
Elemento 1: 75
Elemento 75 foi inserido com sucesso.
Elemento 2: 19
Elemento 19 foi inserido com sucesso.
Elemento 3: 77
Elemento 77 foi inserido com sucesso.
Elemento 4: 14
Elemento 14 foi inserido com sucesso.

ENTER para continuar...
```

```
>> INSERCAO
Quantidade de Elementos: 1
Elemento 1: 75
Elemento 75 ja existe na arvore.

ENTER para continuar..._
```

( EXCLUINDO OS ELEMENTOS ):

Após a inserção, o usuário esta possibilitado em remover os elementos, se necessário. Caso o elemento exista, uma mensagem de sucesso aparecerá na tela. Caso contrário, o programa irá alertar o usuário que o valor é inexistente.

```
>> REMOCAO
Valor a ser Removido: 2
Elemento nao existe!

ENTER para continuar...
```

```
>> REMOCAO
Valor a ser Removido: 14
Elemento 14 removido com sucesso.

ENTER para continuar..._
```

( BUSCANDO ELEMENTO ):

O usuário também poderá procurar um valor. Caso o valor exista, uma mensagem de sucesso na busca será exibida "Elemento encontrado". Caso contrário, o programa irá alertar o usuário que o elemento não existe.

```
>> BUSCA
Valor procurado: 2
Elemento nao existe!

ENTER para continuar..._
```

```
>> BUSCA
Valor procurado: 75
Elemento encontrado

ENTER para continuar...
```

( IMPRESSÃO DOS ELEMENTOS )

Por fim, se o usuário escolher a opção de impressão dos elementos, o programa irá imprimir os elementos dessa

forma:

Impressão em ordem:

Impressão Pre Ordem:

Impressão Pos Ordem:

Impressão Labelled Bracketing:

```
>> IMPRESSAO
Impressao em Ordem:
14 19 75 77

Impressao Pre Ordem:
77 19 14 75

Impressao Pos Ordem:
14 75 19 77

Impressao em Labelled Bracketing:
[[77 [[19 [[14 [] [] ]]]75 [] [] ]]][] ]

ENTER para continuar...
```

>> PARTE 2:

A opção "2" ( Arvore Genealogica ), irá levar para um menu secundário ( IMAGEM 2 ), que trará as opções de:

```
1- Operacoes Basicas em Arvores
2- Arvore Genealogica
0- Sair
```

Inserção e Impressão:

O usuário então, deverá, necessariamente, escolher a opção " 1 " do menu secundário para conseguir trabalhar de forma adequada.

O Usuário deverá escolher a quantidade de elementos a ser inseridos. NOTA: Nesse caso, os elementos serão inseridos 3 a 3. Ou seja, se o usuário escolher inserir 4 elementos, um total de 12 elementos deverá ser inserido. Exemplo:

( INSERINDO ELEMENTOS ):

Na subopção 1:

O Usuário deverá inserir os elementos 3 a 3, como na imagem a seguir.

```
>> INSERCAO
Quantidade de Elementos: 4
(filho, pai, mae): filho1 pai1 mae1
Elemento filho1 foi inserido com sucesso
Elemento pai1 foi inserido com sucesso
Elemento mae1 foi inserido com sucesso
(filho, pai, mae): pai1 avo1 avoh1
Elemento avo1 foi inserido com sucesso
Elemento avoh1 foi inserido com sucesso
(filho, pai, mae): mae1 avo2 avoh2
Elemento avo2 foi inserido com sucesso
Elemento avoh2 foi inserido com sucesso
(filho, pai, mae): avo1 bisavo1 bisavoh1
Elemento bisavo1 foi inserido com sucesso
Elemento bisavoh1 foi inserido com sucesso

ENTER para continuar..._
```

Caso o elemento exista, ele não será adicionado. Caso contrário, ele será adicionado e uma mensagem de inserção aparecerá na tela.

( IMPRIMINDO OS ELEMENTOS):

Na subopção 2:

Após o usuários **ter** inserido os elementos da arvore, então ele poderá imprimir. Nessa opção, a impressão será mostrada em Labelled Bracketing e a Impressão da família por geração

```
>> IMPRIMIR

>>> Impressao em Labelled Bracketing
[filho1 [pai1 [avo1 [bisavo1 ][bisavoh1 ]][avoh1 ]][mae1 [avo2 ][avoh2 ]]]

>>> Impressao da familia por geracao
filho1 pai1 mae1 avo1 avoh1 avo2 avoh2 bisavo1 bisavoh1

ENTER para continuar..._
```

## # MODO DE FUNCIONAMENTO

A struct é composto de uma variável INT para o valor do elemento inserido (caso possua), uma string nome (caso possua) e dois ponteiros que irão levar para os elementos descendentes a esquerda e a direita

```
struct arvore{
    int valor;
    char nome[30];
    struct arvore *esq;
    struct arvore *dir;
};
```

A função "void insercao(struct arvore \*\*raiz, int elemento)" é a função responsável por adicionar os elementos.

Primeiramente, há uma condição verificando se o ponteiro que entra como referência não é nulo. Se ele for nulo,

```
void insercao(struct arvore **raiz, int elemento){
    if(*raiz == NULL ){
        struct arvore *aux = (struct arvore *) malloc(sizeof(struct arvore));
        aux->valor = elemento;
        aux->dir = aux->esq = NULL;
        *raiz = aux;
        printf("Elemento %d foi inserido com sucesso.\n", elemento);
        return;
    }

    else if(elemento < (*raiz)->valor){
        insercao(&(*raiz)->esq, elemento);
        return;
    }

    else if(elemento > (*raiz)->valor){
        insercao(&(*raiz)->dir, elemento);
        return;
    }
    printf("Elemento %d ja existe na arvore.\n", elemento);
}
```

uma variável auxiliar será criada para inserir o primeiro elemento. Se o ponteiro não for nulo, uma comparação será feita entre o elemento que entra por referência e o valor que aponta para o ponteiro da raiz que entra como referência. Se ele for menor, a função entrará no modo recursivo para o ponteiro do lado esquerdo da estrutura.

Se ele for maior, a função entrará no modo recursivo para o ponteiro do lado direito da estrutura. Caso contrário, o elemento já existe e uma mensagem

```
struct arvore *DoisFilhos(struct arvore *raiz){
    if(raiz==NULL)
        return NULL;
    else if(raiz->esq == NULL)
        return raiz;
    else
        return DoisFilhos(raiz->esq);
}
```

A função "struct arvore \*DoisFilhos(struct arvore \*raiz)" servirá como auxiliar da função remoção. Ela irá buscar os possíveis filhos de um lugar na árvore.

A função "void remocao(struct arvore \*\*raiz, int elemento)" é a função que irá buscar um elemento e removerá ele da árvore, caso exista.

A função, basicamente, irá comparar se o elemento de entrada do usuário é maior que o valor do primeiro elemento da árvore, e então entrará na sua forma recursiva, até que o elemento descendente a esquerda e a direita sejam diferentes de nulos, e nesse caso, a função Doisfilhos entrará em ação. Caso contrário, um ponteiro auxiliar será criado e o elemento será removido.

```
void remocao(struct arvore **raiz, int elemento){

    if(elemento < (*raiz)->valor)
        remocao(&(*raiz)->esq, elemento);

    else if(elemento > (*raiz)->valor)
        remocao(&(*raiz)->dir, elemento);

    else if((*raiz)->esq!=NULL && (*raiz)->dir!=NULL){
        struct arvore *aux = NULL;
        aux = DoisFilhos((*raiz)->dir);
        (*raiz)->valor = aux->valor;
        remocao(&(*raiz)->dir, (*raiz)->valor);
    }
    else{
        struct arvore *aux = (*raiz);
        if((*raiz)->esq == NULL)
            (*raiz) = (*raiz)->dir;
        else{
            *raiz = (*raiz)->esq;
        }
        free(aux);
        printf("Elemento %d removido com sucesso.\n", elemento);
    }
}
```

A função "int busca(struct arvore \*raiz, int elemento )", irá buscar um elemento de entrada do usuário. A função irá conferir se o ponteiro é nulo. Se for, ele retornará -1. Se o valor do ponteiro for igual ao elemento digitado pelo usuário, então, a função retornará o mesmo valor digitado pelo usuário. Caso contrario, ele ira comparar se o valor é menor ou maior. Caso for menor ele entrará na forma recursiva para esquerda. Caso for maior, a função entrará na forma recursiva a esquerda.

```
int busca(struct arvore *raiz, int elemento ){
    int x;
    if(raiz==NULL)
        return -1;
    x = raiz->valor;

    if( x == elemento ) return x;
    else if(elemento < x) return busca(raiz->esq, elemento);
    else return busca(raiz->dir, elemento);

}
```

As funções "void imprime\_preordem(struct arvore \*raiz)", "void imprime\_posordem(struct arvore \*raiz)", "void imprime\_emordem(struct arvore \*raiz)" e "void imprime\_labelledbracketing(struct arvore \*raiz)" irá imprimir os valores da arvore, conforme foi inserido pelo usuário. Também de forma recursiva, até que o ponteiro seja nulo.

```
void imprime_preordem(struct arvore *raiz){
    if(raiz==NULL)
        return;
    printf("%d ",raiz->valor);
    imprime_preordem(raiz->esq);
    imprime_preordem(raiz->dir);
}
```

```
void imprime_posordem(struct arvore *raiz){
    if(raiz==NULL)
        return;
    imprime_posordem(raiz->esq);
    imprime_posordem(raiz->dir);
    printf("%d ", raiz->valor);
}
```



```

void imprime_emordem(struct arvore *raiz){
    if(raiz==NULL)
        return;

    imprime_emordem(raiz->esq);
    printf("%d ",raiz->valor);
    imprime_emordem(raiz->dir);
}

void imprime_labelledbracketing(struct arvore *raiz){

    if(raiz==NULL){
        printf("[ ] ");
        return;
    }
    printf("[%d ", raiz->valor);
    imprime_labelledbracketing(raiz->esq);
    imprime_labelledbracketing(raiz->dir);
    printf("]");
}

```

A função "void inserir\_membro(struct arvore \*\*raiz, char string1[], char string2[], char string3[], int valor )", trabalhará de forma semelhante a função "void insercao(struct arvore \*\*raiz, int elemento)" da parte 1. A diferença que ela possuirá 3 entradas strings e irá trabalhar com as mesmas. O valor que entrará como referência, é utilizado para seguir a ordem, dado ao fato que uma string não pode seguir uma ordem como com números.

```

void inserir_membro(struct arvore **raiz, char string1[], char string2[], char string3[], int valor ){
    if(*raiz == NULL ){
        struct arvore *aux = (struct arvore *) malloc(sizeof(struct arvore));
        strcpy(aux->nome, string1);
        aux->dir = aux->esq = NULL;
        aux->valor = valor;
        *raiz = aux;
        printf("Elemento %s foi inserido com sucesso \n", string1);
        if((string2 == NULL) || (string3 == NULL) ){return;}
    }

    if(strcmp(string1, (*raiz)->nome) == 0){
        valor+=1;
        inserir_membro(&(*raiz)->esq, string2, NULL, NULL, valor);
        inserir_membro(&(*raiz)->dir, string3, NULL, NULL, valor);
    }
    else{
        valor+=1;
        if((*raiz)->esq != NULL){
            inserir_membro(&(*raiz)->esq, string1, string2, string3, valor);
        }
        if((*raiz)->dir != NULL){
            inserir_membro(&(*raiz)->dir, string1, string2, string3, valor);
        }
    }
}

```