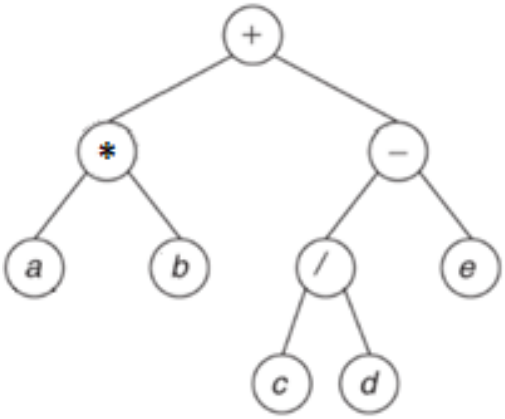
# TRABALHO ACADÊMICO – AV2

## EXERCÍCIO 1

Uma das aplicações de árvores binárias é a representação de expressões aritméticas, onde os operadores ocupam os nós interiores e os operandos ocupam as folhas. Dependendo do tipo de percurso empregado em uma árvore de expressão, o resultado obtido é a expressão descrita em uma determinada notação. Por exemplo: o resultado do percurso em pré-ordem é a expressão aritmética em notação pré-fixa; para o percurso em pós-ordem, o resultado é a expressão em notação pós-fixa e para o percurso em ordem simétrica, o resultado é a expressão em notação infixa. No entanto, diferentemente do que acontece com as notações pré-fixa e pós-fixa, na descrição em notação infixa é necessário "parentizar" a expressão para que as possíveis alterações de precedências dos operadores sejam devidamente respeitadas.

Diante disso, escreva um programa em C que apresente o resultado de um percurso em ordem simétrica (com "parentização") quando aplicado à seguinte árvore de expressão:




As instruções detalhadas são as seguintes:

**1 -** descompacte e abra no Code Blocks o projeto `exemplo_1` disponível no AVA, no *card* MIDIATECA DA DISCIPLINA, seção "Árvores Binárias", arquivo "Implementação - Árvores Binárias.rar";

**2 -** altere o exemplo de árvore binária existente no arquivo `main.c` para que represente a árvore binária apresentada na descrição do exercício;

**3 -** implemente, no próprio arquivo `main.c`, a função `percorre( )` com as seguintes instruções:

```
percorre(raiz da árvore)
   se (subárvore esquerda é não nula) então
      escreva('(');
      percorre(subárvore esquerda);
   fimse;
   escreva(caractere);
   se (subárvore direita é não nula) então
      percorre(subárvore direita);
      escreva(')');
   fimse;
```

**4 -** execute a função `percorre( )` passando como parâmetro a raiz da árvore criada.

Após a execução do programa, copie todo o conteúdo exibido na tela em um arquivo texto com o nome `resultados.txt`.

## EXERCÍCIO 2

As redes de comunicação de dados adotam uma estratégia bastante interessante para transmitir um fluxo de informações de um ponto a outro. Devido às eventuais perdas que acontecem durante a transmissão, essa informação é fragmentada em pacotes de dados, de forma que cada pacote carrega parte da informação e também um número de identificação. Como a ordem de chegada dos pacotes ao destino pode não ser a mesma de partida da origem, essa identificação permite que os fragmentos sejam reorganizados na sequência correta quando chegam ao seu destino e, com isso, tem-se o fluxo de informação devidamente reconstruído. Durante a transmissão, alguns desses pacotes podem ser perdidos e, como não é possível prever quais serão perdidos, cada pacote é enviado várias vezes para que as eventuais perdas sejam reparadas. Por outro lado, caso não ocorra a perda, o mesmo pacote chegará várias vezes ao destino. Assim, no momento em que o fluxo de informação for reconstruído, é necessário descartar os pacotes duplicados.

A implementação dessa estratégia usa como estrutura de dados básica uma **árvore binária de busca** situada no destino. Dessa forma, à medida que os pacotes chegam ao destino, são **inseridos** em uma árvore binária de busca considerando como chave o número de identificação do pacote. Essa mesma operação de inserção vai garantir que pacotes repetidos não possam ser inseridos na árvore. Uma vez inseridos todos os pacotes na árvore, para a reconstrução do fluxo de informação, basta **remover** os pacotes da árvore em ordem crescente de acordo com as suas chaves.

O exemplo a seguir ilustra o funcionamento de todo o processo:

Mensagem a ser enviada:

**ALÔ MUNDO!**

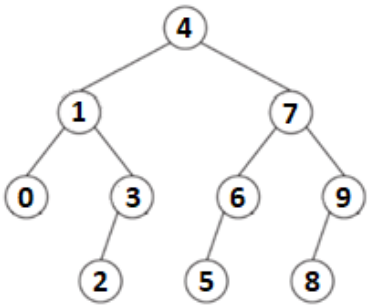
Fragmentação da mensagem:

| **id** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| **dado** | **‘A’** | **‘L’** | **‘Ô’** | **‘ ’** | **‘M’** | **‘U’** | **‘N’** | **‘D’** | **‘O’** | **‘!’** |

Ordem de chegada dos pacotes (já considerando a replicação) ao destino:

| **id** | 4 | 1 | 7 | 1 | 0 | 3 | 2 | 7 | 6 | 5 | 9 | 2 | 5 | 8 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **dado** | **‘M’** | **‘L’** | **‘D’** | **‘L’** | **‘A’** | **‘ ’** | **‘Ô’** | **‘D’** | **‘N’** | **‘U’** | **‘!’** | **‘Ô’** | **‘U’** | **‘O’** |

Árvore binária de busca correspondente (os nós representam as chaves e as chaves duplicadas foram descartadas):




Mensagem reconstruída (após a remoção dos nós em ordem crescente):

**ALÔ MUNDO!**

Diante do exposto, implemente um programa em C que simule essa estratégia de transmissão de dados. As instruções detalhadas são as seguintes:

**1 -** descompacte e abra no Code Blocks o projeto `exemplo_2` disponível no AVA, no *card* MIDIATECA DA DISCIPLINA, seção “Árvores Binárias de Busca”, arquivo “Implementação - Árvores Binárias de Busca.rar”;

**2 -** modifique o arquivo `main.c` para executar as seguintes operações:

**2.1 -** solicite ao usuário que entre com uma mensagem a ser transmitida pela rede;

**2.2 -** imprima a mensagem;

**2.3 -** fragmente a mensagem de forma que cada caractere represente um pacote a ser transmitido, lembrando que a cada pacote deve ser associado um número de identificação;

**2.4 -** para cada pacote, gere um número aleatório inteiro entre 1 e 10, que vai determinar a quantidade de vezes que o pacote deve ser replicado;

**2.5 -** gere uma nova sequência de pacotes em que estejam embaralhados tanto os pacotes da mensagem original quanto os pacotes replicados;

**2.6 -** imprima a nova sequência de pacotes;

**2.7 -** construa a árvore binária de busca a partir da nova sequência de pacotes;

**2.8 -** imprima a árvore;

**2.9 -** reconstrua a mensagem original a partir da árvore binária de busca.

**2.10 -** imprima a mensagem.

Após a execução do programa, copie todo o conteúdo exibido na tela no mesmo arquivo `resultados.txt` do exercício anterior.

## OBSERVAÇÕES SOBRE A ENTREGA E A AVALIAÇÃO:

- Os projetos com os TAD's fornecidos para ambos os exercícios devem ser obrigatoriamente utilizados;
- Devem ser postados no AVA apenas o arquivo `main.c` de cada exercício e o arquivo `resultados.txt`. O arquivo do exercício 1 deve ser nomeado como `main_1.c` e o do exercício 2 como `main_2.c`.
- Para cada exercício, o arquivo `main.c` correspondente será testado no projeto original, por isso, não se deve fazer alterações em quaisquer outros arquivos do projeto para não interferir no funcionamento do programa.
- Caso o arquivo `main.c` de um exercício não funcione no projeto original correspondente, a nota desse exercício será **0,0**.
- A postagem dos arquivos deve acontecer, **impreterivelmente**, até o dia **18/11/20** às **23:55**.