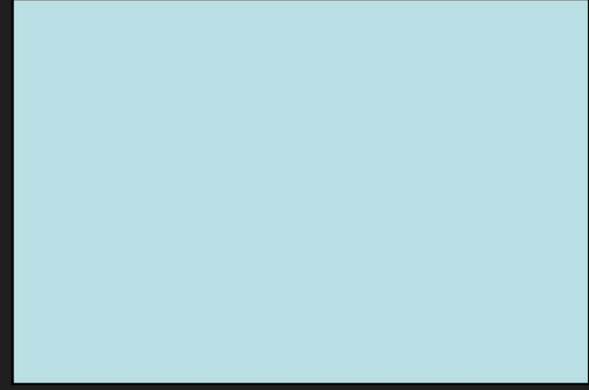
Pilha - Alocação Encadeada

Com base no que foi visto implemente a operação create() que compõem o TAD PILHA_ENC.

```
 1  typedef struct nodo
 2  {
 3      int inf;
 4      struct nodo * next;
 5  }NODO;
 6  typedef NODO * PILHA_ENC;
 7  void create (PILHA_ENC *);
 8  int is_empty (PILHA_ENC);
 9  void push (PILHA_ENC *, int);
10  int top (PILHA_ENC);
11  void pop (PILHA_ENC *);
12  int top_pop (PILHA_ENC *);
13  void destroy (PILHA_ENC);
```

```
1   void create (PILHA_ENC *pp)
2   {
3       *pp=NULL;
4   }
```



Pilha - Alocação Encadeada

Com base no que foi visto implemente a operação is_empty() que compõem o TAD PILHA_ENC.

```
 1  typedef struct nodo
 2  {
 3      int inf;
 4      struct nodo * next;
 5  }NODO;
 6  typedef NODO * PILHA_ENC;
 7  void create (PILHA_ENC *);
 8  int is_empty (PILHA_ENC);
 9  void push (PILHA_ENC *, int);
10  int top (PILHA_ENC);
11  void pop (PILHA_ENC *);
12  int top_pop (PILHA_ENC *);
13  void destroy (PILHA_ENC);
```

UNIVASF

```
1   int is_empty (PILHA_ENC p)
2   {
3       return (!p);
4   }
```

Pilha - Alocação Encadeada

Com base no que foi visto implemente a operação push() que compõem o TAD PILHA_ENC.

```
 1  typedef struct nodo
 2  {
 3      int inf;
 4      struct nodo * next;
 5  }NODO;
 6  typedef NODO * PILHA_ENC;
 7  void create (PILHA_ENC *);
 8  int is_empty (PILHA_ENC);
 9  void push (PILHA_ENC *, int);
10  int top (PILHA_ENC);
11  void pop (PILHA_ENC *);
12  int top_pop (PILHA_ENC *);
13  void destroy (PILHA_ENC);
```

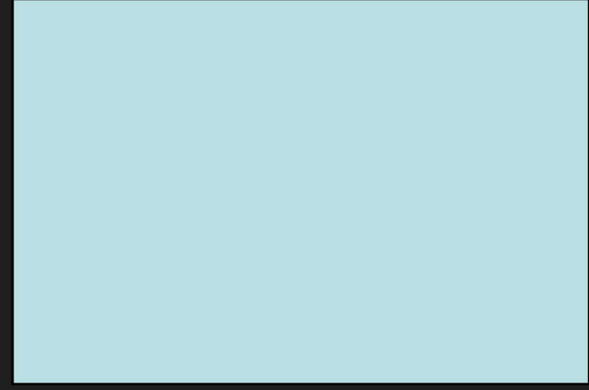
```c
void push (PILHA_ENC *pp, int v)
{
    NODE *new;
    new = (NODE *) malloc (sizeof(NODE));
    if (!new)
    {
        printf ("\nERRO! Memoria insuficiente!\n");
        exit (1);
    }
    new->inf = v;
    new->next = *pp;
    *pp=new;
}
```

Pilha - Alocação Encadeada

Com base no que foi visto implemente a operação top() que compõem o TAD PILHA_ENC.

```
1   typedef struct nodo
2   {
3       int inf;
4       struct nodo * next;
5   }NODO;
6   typedef NODO * PILHA_ENC;
7   void create (PILHA_ENC *);
8   int is_empty (PILHA_ENC);
9   void push (PILHA_ENC *, int);
10  int top (PILHA_ENC);
11  void pop (PILHA_ENC *);
12  int top_pop (PILHA_ENC *);
13  void destroy (PILHA_ENC);
```

```c
int top (PILHA_ENC p)
{
    if (!p)
    {
        printf ("\nERRO! Consulta em pilha vazia!\n");
        exit (2);
    }
    return (p->inf);
}
```



Pilha - Alocação Encadeada

Com base no que foi visto implemente a operação pop() que compõem o TAD PILHA_ENC.

```
1   typedef struct nodo
2   {
3       int inf;
4       struct nodo * next;
5   }NODO;
6   typedef NODO * PILHA_ENC;
7   void create (PILHA_ENC *);
8   int is_empty (PILHA_ENC);
9   void push (PILHA_ENC *, int);
10  int top (PILHA_ENC);
11  void pop (PILHA_ENC *);
12  int top_pop (PILHA_ENC *);
13  void destroy (PILHA_ENC);
```

```c
void pop (PILHA_ENC *pp)
{
    if (!(*pp))
    {
        printf ("\nERRO! Retirada em pilha vazia!\n");
        exit (3);
    }
    else
    {
        NODE *aux=*pp;
        *pp=(*pp)->next;
        free (aux);
    }
}
```

Pilha - Alocação Encadeada

Com base no que foi visto implemente a operação top_pop() que compõem o TAD PILHA_ENC.

```
 1  typedef struct nodo
 2  {
 3      int inf;
 4      struct nodo * next;
 5  }NODO;
 6  typedef NODO * PILHA_ENC;
 7  void create (PILHA_ENC *);
 8  int is_empty (PILHA_ENC);
 9  void push (PILHA_ENC *, int);
10  int top (PILHA_ENC);
11  void pop (PILHA_ENC *);
12  int top_pop (PILHA_ENC *);
13  void destroy (PILHA_ENC);
```

```c
int top_pop (PILHA_ENC *pp)
{
    if (!(*pp))
    {
        printf ("\nERRO! Consulta e retirada em pilha vazia!\n");
        exit (4);
    }
    else {
        int v=(*pp)->inf;
        NODE *aux=*pp;
        *pp=(*pp)->next;
        free (aux);
        return (v);
    }
}
```

Pilha - Alocação Encadeada

Com base no que foi visto implemente a operação destroy() que compõem o TAD PILHA_ENC.

```
 1  typedef struct nodo
 2  {
 3      int inf;
 4      struct nodo * next;
 5  }NODO;
 6  typedef NODO * PILHA_ENC;
 7  void create (PILHA_ENC *);
 8  int is_empty (PILHA_ENC);
 9  void push (PILHA_ENC *, int);
10  int top (PILHA_ENC);
11  void pop (PILHA_ENC *);
12  int top_pop (PILHA_ENC *);
13  void destroy (PILHA_ENC);
```

```c
void destroy (PILHA_ENC l)
{
    PILHA_ENC aux;
    while (l)
    {
        aux = l;
        l = l->next;
        free(aux);
    }
}
```

Pilha - Alocação Encadeada

O mesmo que discutimos a respeito das filas ocorre com as pilhas.

Ou seja, uma pilha nada mais é do que uma lista com uma disciplina de acesso.

Logo, podemos nos utilizar de todos os conceitos vistos em listas para implementarmos pilhas.

Por exemplo, podemos utilizar uma lista encadeada com nó cabeçalho (contendo o número de elementos) para armazenar uma pilha.

UNIVASF

Aplicação de Pilhas

Notações: in, pré e posfixada

Examinaremos agora uma importante aplicação que ilustra a utilização do TAD pilha.

Considerando a soma de **A** mais **B**.

Aplicação do **operador** “+” sobre os **operandos A** e **B.**

A + B (representação infixada).

Existem duas notações alternativas.

Notações: in, pré e posfixada

São elas:

+ A B **prefixada**

A B + **posfixada**

Analisando expressões infixadas um pouco mais complexas, como, por exemplo, **A + B * C**.

Notamos a necessidade da definição de precedência entre os operadores (em casos em que é preciso alterar a ordem de precedência pré estabelecida se utilizam parênteses) visando eliminar a ambiguidade, tornando a tarefa computacional menos simples.

UNIVASF

Notações: in, pré e posfixada

A representação pré e posfixada para expressões aritméticas são mais conveniente do ponto de vista computacional.

Para ilustrarmos os diferentes tipos de representações, utilizaremos em nosso exemplos cinco operações binárias: adição, subtração, multiplicação, divisão e exponenciação.

Veremos alguns exemplos de expressões representadas nas três formas.

UNIVASF

Notações: in, pré e posfixada

Forma Infixada	Forma Prefixada	Forma Posfixada
A + B - C	- + ABC	AB + C -
(A + B) * (C - D)	*+ AB – CD	AB + CD - *
A ^ B * C - D + E / F / (G + H)	+ - * ^ ABCD / / EF + GH	AB ^ C * D - EF / GH + / +
((A + B) * C - (D - E)) ^ (F + G)	^ -* + ABC - DE + FG	AB + C * DE - - FG + ^
A - B / (C * D ^ E)	- A / B * C ^ DE	ABCDE ^ * / -

Notações: in, pré e posfixada

Vamos agora implementar uma função para avaliar uma expressão aritmética posfixada.

Devemos inicialmente definir a entrada.

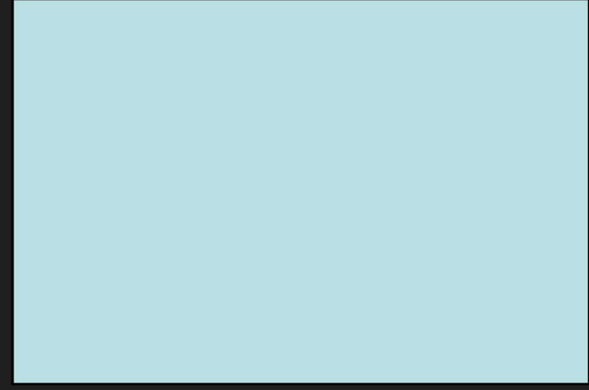
Sendo assim, vamos imaginar uma string representando a expressão aritmética posfixada.

Vamos trabalhar com operandos positivos e compostos por apenas um dígito.

```c
int avaliar (char *e)
{
    char symbol;
    int i=0;
    PILHA_ENC pilha_operandos;
    create (&pilha_operandos);
    while (symbol = e[i++])
        if (eh_operando(symbol))
            push (&pilha_operandos, symbol-'0');
        else
        {
            int op2=top_pop(&pilha_operandos),
            op1=top_pop(&pilha_operandos);
            push (&pilha_operandos, aplicar (op1, symbol, op2));
        }
    return (top_pop(&pilha_operandos));
}
```

```c
int eh_operando(char op)
{
    return (op != '+' && op != '-' &&
    op != '*' && op != '/' && op != '^');
}
```

```
1  int aplicar (int operando1, char operador,
2  int operando2) {
3      switch (operador)
4      {
5          case '+': return (operando1 + operando2);
6          case '-': return (operando1 - operando2);
7          case '*': return (operando1 * operando2);
8          case '/': return (operando1 / operando2);
9          case '^': return ((int)pow(operando1, operando2));
10     }
11 }
```



```c
int main()
{
    char expr [MAXCOLS];
    int position = 0;
    while ((expr[position++] = getchar ()) != '\n');
    expr[--position]='\0';
    printf ("%s%s","a expressão posfixada original eh ",expr);
    printf (" = %d\n", avaliar(expr));
    return 0;
}
```

Notações: in, pré e posfixada

Já implementamos uma função para avaliar expressões posfixadas, agora, vamos tratar o problema de converter uma expressão infixada em uma posfixada.

Inicialmente, trataremos apenas a questão da precedência, deixando a questão da utilização de parênteses para uma análise posterior.

Defina uma função simples, cujo objetivo é verificar a precedência entre operadores.

int prcd (char op1, char op2) ;

UNIVASF

```c
int prcd (char op1, char op2)
{
    if ((op1=='+' || op1=='-') && (op2=='+' ||
    op2=='-'))
        return (1);
    if ((op1=='*' || op1=='/') && (op2=='+' ||
    op2=='-' || op2=='*' || op2=='/'))
        return (1);
    if (op1=='^' && (op2=='^' || op2=='+' ||
    op2=='-' || op2=='*' || op2=='/'))
        return (1);
    return (0);
}
```

Notações: in, pré e posfixada

Agora, implemente uma função que converte uma string infixada sem parênteses em uma string posfixada.

void converter_s_p(char *o, char *d);

```c
void converter_s_p(char *o, char *d) {
    char symbol;
    int i1=0, i2=0;
    PILHA_ENC opstk;
    cria_pilha (&opstk);
    while (symbol=o[i1++])
        if (eh_operando(symbol))
            d[i2++]=symbol;
        else {
            while (!eh_vazia(opstk) && prcd (top(opstk),symbol))
                d[i2++]=top_pop(&opstk);
            push(&opstk, symbol);
        }
    while (!eh_vazia(opstk))
        d[i2++]=top_pop(&opstk);
    d[i2]='\0';
}
```

Notações: in, pré e posfixada

Que alterações seriam necessárias para que o algoritmo anterior seja capaz de converter uma string infixada com parênteses em uma string posfixada?

São necessárias apenas pequenas alterações.

Quando um ‘**(**’ for lido, deverá ser empilhado.

Quando um ‘**)**’ for lido, todos os operadores até o primeiro ‘**(**’ deverão ser retirados da pilha e inseridos na string posfixada.

Notações: in, pré e posfixada

Quando esses operadores forem removidos da pilha e o parêntese de abertura for descoberto, uma ação especial deve ser tomada: o parêntese de abertura deve ser removido da pilha e descartado, juntamente com o parêntese de fechamento, em vez de ser colocado na string posfixada ou na pilha.

Com estas pequenas alterações podemos fazer uma função para converter uma string infixada qualquer em uma string posfixada.

UNIVASF

Notações: in, pré e posfixada

Com base no que vimos, construa um programa, na linguagem C, que leia da entrada padrão uma string representando uma expressão infixada, com presença de parênteses, a converta em posfixada e a avalie retornando na saída padrão o resultado da avaliação.