Ejercicios Pilas

Ejercicio 1. Balanceo de paréntesis. Implementa con control de errores el algoritmo de balanceo de paréntesis:

```
Bool balancedExpression(char *str){
  Stack *s = NULL;
  Status st = OK;
  int i;

  if (!str) return FALSE;

  s = stack_init();
  if (!s) return FALSE;

  for (i=0, str[i] != '\0' && st == OK, i++){
    if (str[i] == '(')
      st = stack_push(s, str[i]);
    else if (str[i] == ')' && !stack_isEmpty(s))
      st = (stack_pop(s) != NULL);
    else if (str[i] == ')' && stack_isEmpty(s))
      st = ERROR;
  }

  if (!st || !stack_isEmpty(s)){
    while(!stack_isEmpty(s))
      stack_pop(s);
    stack_free(s);
    return FALSE;
  }

  stack_free(s);
  return TRUE;
}
```

Ejercicio 2. Inversión de pila. Implementa una función que invierta el contenido de una pila. La pila no debe modificarse cuando se devuelva ERROR:

```
Status stackInverter (Stack **s){
  Stack *new = NULL;
  Status st = OK;
  void *elem = NULL;

  if (!s) return ERROR;

  new = stack_init();
  if (!new) return ERROR;

  while(st == OK && (elem = stack_pop(*s)) != NULL)
    st = stack_push(new, elem);

  if (!st){
    stack_push(*s, elem);
    while ((elem = stack_pop(new)) != NULL)
      stack_push(*s, elem);
    stack_free(new);
    return ERROR;
  }

  stack_free(*s);
  *s = new;

  return OK;
}
```

Ejercicio 3. Merge stacks. Implementa una función que devuelva una pila ordenada en orden ascendiente a partir de dos pilas ordenadas de la misma forma:

```
Status mergeStacks(Stack *s1, Stack *s2, Stack *s3, int(*ele_cmp)(void*,void*)){
  void *e1 = NULL, *e2 = NULL;
  Status *st = OK;


  if (!s1 || !s2 || !s3) return ERROR;

  while (!stack_isEmpty(s1) && !stack_isEmpty(s2) && st == OK){
    if (ele_cmp(stack_top(s1), stack_top(s2)) >= 0){
      st = stack_push(s3, stack_pop(s1));
    else{
      st = stack_push(s3, stack_pop(s2));
  }

  while (!stack_isEmpty(s1) && st == OK)
    st = stack_push(s3, stack_pop(s1));

  while (!stack_isEmpty(s2) && st == OK)
    st = stack_push(s3, stack_pop(s2));

  return st;
}
```

Ejercicio 4. Invertir cadenas. Implementa una función con control de errores que, usando una pila, invierta una cadena de caracteres:

```
char *string_invert(char *str){
  Stack *s = NULL;
  Status st = OK;
  char *new = '\0';
  int i, size;

  if (!str) return NULL;

  s = stack_init();
  if (!s) return NULL;

  for (size=0; str[size] != '\0' && st == OK; size++){
    st = stack_push(s, str[size]);
  }

  for (i=0; i<size && st == OK; i++){
    new[i] = stack_pop(s);
  }

  stack_free(s);
  return new;
}
```

Ejercicio 5. Copia de pila. Implementa una función que realice un duplicado de una pila. La función no debe modificar la pila original:

```
Stack *stack_deepCopy(Stack *s, void*(*ele_copy)(const void*), void(*ele_free)(void*)){
  Stack *new = NULL, *aux = NULL;
  Status st = OK;
  void **elems, *e;
  int i;

  if (!s || !ele_cpy || !ele_free) return NULL;
```

```
new = stack_init();
if (!new) return NULL;

aux = stack_init();
if (!aux){
  stack_free(new);
  return NULL;
}

elems = malloc(sizeof(void*));
if (!elems){
  stack_free(new);
  stack_free(aux);
  return NULL;
}
while (!stack_isEmpty(s) && st == OK){
  e = stack_pop(s);
  st = stack_push(aux, e);
}

for (i=0, !stack_isEmpty(aux) && st == OK, i++){
  e = stack_pop(aux);
  stack_push(s, e);
  elems[i] = ele_cpy(e);
  st = stack_push(new, elems[i]);
  realloc(elems, (i+1)*sizeof(void*));
  if (!elems)
    st = ERROR;
}

if (st == ERROR){
  stack_push(s, e);
  while (!stack_isEmpty(aux))
    stack_push(s, stack_pop(aux));
  stack_free(new);
}

stack_free(aux);
stack_free(elems);

return st;
```