Algoritmos

Linguagem C - Ponteiros e alocação dinâmica

Lista de exercícios

1. Sobre o programa a seguir:

```
int main(){
  int a=10,b=20;
  int *pa, *pb;
  pa = &a;
  pb = &b;
  *pa = *pb * 2;
  printf("%d,%d\n",a,b);
  return 0;
}
```

   Determine o que o programa escreve na saída padrão. Explique, usando desenho da memória, como você chegou a essa conclusão.

2. Sobre o programa a seguir:

```
int main() {
  int a = 10, b = 20, c = 30;
  int *ponteiro;
  ponteiro = &b;
  ponteiro = ponteiro + 1;
  *ponteiro = 50;
  printf("a,b,c =  %d, %d, %d\n",a,b,c);
  return 0;
}
```

   Determine o que o programa escreve na saída padrão. Explique, usando desenho da memória, como você chegou a essa conclusão.

3. Escreva uma função que receba como parâmetro dois ponteiros para inteiros e realize uma troca de valores. O conteúdo do endereço apontado por a deve ir para o endereço apontado por b e o conteúdo do endereço apontado por b deve ir para o endereço apontado por a.

```
void swap (int *a, int *b);
```

4. Seja o trecho de código a seguir:

```
int main(){
  int x;
  int *ptr = &x;
  /* ...*/
  return 0;
}
```

   A forma correta de ler um valor inteiro da entrada padrão e armazenar na variável x é:

   (a) `scanf("%d",x);`

   (b) `scanf("%d",*x);`

   (c) `scanf("%d",ptr);`

   (d) `scanf("%d",*ptr);`

5. Considere o código a seguir:

array_list1.c

```c
#include <stdio.h>
#include <stdlib.h>

struct array_list{
  int *data;
  unsigned int size;
  unsigned int capacity;
};

void array_list_increase_capacity(struct array_list *list){
  /* TODO*/
  fprintf(stderr,"Error on memory allocation.\n");
  exit(-1);
}

struct array_list * array_list_create(){
  struct array_list *new_list;
  new_list = (struct array_list*)malloc(sizeof(struct array_list));
  if (new_list==0){ /* Error */
    fprintf(stderr,"Error on memory allocation.\n");
    exit(-1);
  }
  new_list->capacity = 10;
  new_list->size = 0;
  new_list->data = (int*)malloc(sizeof(int)*new_list->capacity);
  if (new_list->data==0){ /* Error */
    fprintf(stderr,"Error on memory allocation.\n");
    exit(-1);
  }
  return new_list;
}

void array_list_read_until(struct array_list *list, int end_reading){
    int x;
    while (scanf("%d",&x),x!=end_reading){
      if (list->size==list->capacity)
        array_list_increase_capacity(list);
      list->data[list->size++] = x;
    }
}

void array_list_print(struct array_list *list){
    printf("[");
    if (list->size>0){
      printf("%d",list->data[0]);
      for (int i=1 ; i<list->size ; ++i)
        printf(", %d",list->data[i]);
    }
    printf("]");
}

int main(){
  struct array_list *list01 = array_list_create();
  array_list_read_until(list01,-1);
  array_list_print(list01);
  printf("\n");
  free(list01->data);
  free(list01);
  return 0;
}
```

Implemente a função `array_list_increase_capacity` para que a mesma aumente a capacidade da lista adicionando mais 10 espaços.

6. A operação de inserir um elemento no final de uma lista é bastante comum em programas que manipulam coleções em forma de lista/vetor. Implemente a função `array_list_append()`, que insere um novo elemento no final da lista, aumentando sua capacidade se necessário. A assinatura da função é

   ```
   void array_list_append(struct array_list *list, int value);
   ```

   Modifique a função `array_list_read_until()`, da questão anterior, para que a mesma use a função `array_list_append()` que você implementou.

7. Implemente a função `list_pop()`, que remove o último elemento da lista.

   ```
   void pop(struct array_list * list);
   ```

   O programa a seguir lê uma sequencia de inserções e remoções a partir da entrada padrão.

array_list2.c

```c
#include <stdio.h>
#include <stdlib.h>

struct array_list{
  int *data;
  unsigned int size;
  unsigned int capacity;
};
/* Implementações de:
   array_list_increase(), array_list_create(), array_list_append() e array_list_print();
*/
void list_pop(struct array_list *list){ /* TODO */ }

int main(){
  struct array_list *list01 = array_list_create();
  int op,value;
  while (scanf("%d",&op,op!=0)){
    if (op==1){ // append
      scanf("%d",&value);
      array_list_append(list01,value);
    } else if (op==2){ // pop
      array_list_pop(list01);
    }
  }
  array_list_print(list01); printf("\n");
  free(list01->data); free(list01);
  return 0;
}
```

Exemplo de entrada e saída

Exemplo de entrada 1	Exemplo de saída 1
1 10 1 20 1 30 1 50 2 1 40 1 50 1 100 1 200 2 2 1 60 0	[10 20 30 40 50 60]

8. Implemente a função a seguir, que realiza uma cópia de uma lista dinâmica de números inteiros e retorna um ponteiro para a nova lista criada.

```
struct array_list * array_list_clone(struct array_list * list);
```

Implemente um programa para testar a função.

9. Escreva uma função que insira um valor inteiro na lista de números inteiros. A função deve retornar um inteiro com a quantidade de números a pós a inserção.

```
unsigned int array_list_insert(struct array_list * list, int value, int index);
```

OBS 1: Verifique se o índice passado como parâmetro é válido.

OBS 2: Todos os elementos, a partir do índice de inserção, devem ser deslocados uma posição "à direita".

10. Escreva uma função que remova um valor de um determinado índice de uma lista.

```
unsigned int array_list_remove(struct array_list * list, int index);
```

A função deve retornar a quantidae de elementos após a remoção.

OBS 1: Verifique se o índice passado como parâmetro é válido.

OBS 2: Todos os elementos, a partir do índice de remoção, devem ser deslocados uma posição "à esquerda".