Instituto Federal de Educação, Ciência e Tecnologia da Paraíba
Campus Campina Grande
Coordenação de Informática

Laboratório de Estrutura de Dados e Algoritmos – Engenharia de Computação
Prof. Ruan Delgado Gomes

Exercício - *Structs* e Alocação Dinâmica de Memória

1 Conceitos Iniciais

Neste exercício vamos praticar os conceitos básicos de struct e alocação dinâmica de memória. Verifique o código definido a seguir e responda as perguntas que vem em seguida:

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

typedef struct {
    int idade;
    char nome[50];
    int *notas;
} aluno;

int main() {
    aluno a1, a2;
    a1.idade = 10;
    strcpy(a1.nome, "Pedro"); //a1.nome = "Pedro" não funciona
    a1.idade = 12;
    strcpy(a1.nome, "Maria");

    a1.notas = (int*)malloc(5*sizeof(int));
    a2.notas = (int*)malloc(4*sizeof(int));

    int i;
    for(i = 0; i < 5; i++) {
        a1.notas[i] = 100;
    }
    a2.notas = a1.notas;
    for(i = 0; i < 4; i++) {
        a2.notas[i] -= 10;
    }
    double media = 0;
    for(i = 0; i < 5; i++) {
        media += a1.notas[i];
    }
    media = media/5;
    printf("Media de %s: %lf\n",a1.nome,media);

    media = 0;
    for(i = 0; i < 4; i++) {
        media += a2.notas[i];
```

```
39      }
40      media = media/4;
41      printf("Media de %s: %lf\n",a2.nome,media);
42
43      free(a1.notas);
44      free(a2.notas);
45
46      return 0;
47  }
```

Questão 1: Qual a saída fornecida por esse programa?

Questão 2: O que ocorre na linha 25?

Questão 3: O que ocorre nas linhas 43 e 44? Devido ao que foi feito anteriormente no código, isso pode gerar algum problema?

Questão 4: Reescreva esse código e corrija os erros para que ele funcione da maneira esperada. Ou seja, deve-se colocar corretamente na saída a média de cada aluno, sabendo que as 4 notas do alunos a2 equivalem às 4 primeiras notas do aluno a1 subtraídas de 10.

2 Função splitInt

Neste exercício, você deve implementar uma função semelhante à função *split* disponível em python. Essa função, denominada *splitInt*, deve receber como parâmetro uma string contendo números inteiros separados por espaços em branco e deve retornar um ponteiro para um array de inteiros construído após a separação dos valores inteiros contidos na string. Por exemplo, se for recebida como parâmetro a string (array de ***char***) "3 24 55 2 0", deve-se gerar um array de inteiros de 5 posições (V = {3, 24, 55, 2, 0}). É importante notar que essa função deve funcionar, mesmo que existam espaços em branco no início ou no final na string, bem como uma quantidade variável de espaços em branco entre os números. Por outro lado, é garantido que todos os números inteiros contidos na string cabem em uma variável do tipo ***int***.

2.1 Função para Verificação e Contagem de Números

Antes de implementar a função *splitInt*, deve-se implementar uma função auxiliar para verificar se a string passada como parâmetro é válida (ou seja, contém apenas números e espaços em branco) e para retornar a quantidade de números independentes contidos na string. Essa função será usada dentro da função *splitInt*.

A função para verificação e contagem, denominada *verifyListInt* deve receber como parâmetro um array de ***char*** e deve retornar a quantidade de números independentes contidos na string. Caso não exista nenhum número na string ou se existir algum caractere inválido, deve-se retornar o valor 0.

2.2 Implementação da Função splitInt

A função splitInt deve receber como parâmetro um array de ***char*** e um valor inteiro **passado por referência**, que receberá o tamanho do array de inteiros que foi gerado a partir da string. A função deve retornar um ***int ****, que deve apontar para um array de inteiro criado dinamicamente a partir dos valores contidos na string.

No início da função, deve-se utilizar a função implementada anteriormente (*verifyListInt*) para obter a quantidade de números contidos na string. Caso a função *verifyListInt* retorne o valor 0, deve-se atribuir o valor 0 ao valor inteiro passado como parâmetro por referência e deve-se retornar NULL.

Caso o valor retornado por *verifyListInt* seja um número maior que zero, deve-se atribuir esse tamanho à variável inteira passada como parâmetro por referência e deve-se alocar um vetor de inteiros utilizando o tamanho retornado por *verifyListInt*. Após isso, deve-se separar os números contidos na string, convertê-los para inteiro (pode usar a função pronta *atoi*) e colocá-los no array que foi criado dinamicamente. Ao final, deve-se retornar o ponteiro para esse novo array criado dinamicamente.

As assinaturas das duas funções a serem criadas são mostradas a seguir:

```
1    int verifyListInt(const char *s);
2    int* splitInt(const char *s, int *size);
```

O código a seguir mostra a função main() de um programa que testa a função *splitInt*.

```
1  int main() {
2      char ent[100];
3      scanf("%[^\n]s", ent);
4      int size ;
5      int *ent_int = split_int(ent,&size);
6      if(ent_int == NULL) {
```

```
7           printf ("String inválida");
8           return 1;
9       }
10      printf("Size: %d\n",size);
11      int i;
12      for(i = 0; i < size; i++) {
13          printf("%d ",ent_int[i]);
14      }
15      return 0 ;
16  }
```

3 Implementando Abstração para um Pacote de Rede

Neste exercício vamos construir abstrações úteis para a implementação de um protocolo de rede. Deve-se criar as abstrações para permitir definir novos tipos relacionados às entidades do problema, bem como funções auxiliares, que permitam analisar as informações dos pacotes trafegados na rede. A definição do novo tipo estruturado que representa o pacote de rede, bem como as funções que manipulam o pacote de rede devem ser escritas em um arquivo denominado *pacote.h*. Após implementar a *struct* e as funções definidas nas próximas sub-seções, escreva um programa de teste (denominado *testepacotes.c*), que declara pacotes e usa todas as funções descritas no arquivo *pacote.h*.

Coloque os dois arquivos em uma mesma pasta e realize a inclusão de *pacote.h* em *testapacotes.c* da seguinte forma: #include "pacote.h".

3.1 Definindo os Elementos da Rede

Um pacote de rede no problema abordado neste exercício é composto por um conjunto de campos, listados a seguir:

- Identificador do pacote: valor inteiro de 8 bits;
- Tipo do pacote: pode ser pacote de dados e pacote de reconhecimento (ACK);
- Endereço de origem: valor inteiro de 32 bits;
- Endereço de destino: valor inteiro de 32 bits;
- Carga útil: array de até 100 bytes (alocado dinamicamente);
- Tamanho: um valor inteiro de 8 bits que contém a quantidade de bytes que a carga útil do pacote possui.

Por exemplo, quando um dispositivo de endereço 10, quer enviar um pacote de dados para outro dispositivo de endereço 20, ele deve enviar um conjunto de bytes pela rede contendo todas as informações do pacote. Nesse caso, o pacote terá um identificador sequencial (o primeiro pacote enviado possui valor 0, o segundo valor 1 etc), um tipo (pode-se usar um inteiro para identificar o tipo), um endereço de origem igual a 10, um endereço de destino igual a 20, um array de até 100 bytes que conterá a informação de fato que o dispositivo quer transmitir, e um valor inteiro que contém a quantidade de bytes de carga útil que o pacote contém.

A primeira parte deste exercício é criar um novo tipo estruturado chamado Packet, que contenha todos os campos descritos anteriormente.

3.2 Escrevendo Funções para Manipulação de Pacotes

Quando um pacote é recebido na rede, ele precisa ser analisado. Neste exercício, não será necesseário implementar as funcionalidades necessárias para permitir receber ou transmitir pacotes de verdade por meio de uma interface de rede. No entanto, vamos criar e testar funções auxiliares para a manipulação dos pacotes de rede. As funções a serem criadas são descritas a seguir:

- Criação do pacote de dados (*createDataPacket*): Deve receber como parâmetro o identificador do pacote, os endereços de origem e destino e a quantidade de bytes de carga útil. Dentro da função deve-se criar um novo pacote de forma dinâmica (usando *malloc()*), deve-se alocar espaço para a sua carga útil, preencher os dados do pacote de acordo com os parâmetros e por fim retornar o endereço de memória do pacote recém criado;

- Criação do pacote de reconhecimento (*createACKPacket*): Deve receber como parâmetro o identificador do pacote, os endereços de origem e destino. Um pacote de reconhecimento não possui carga útil, portanto o ponteiro que aponta para a carga útil deve receber o valor NULL e o atributo que representa a quantidade de bytes da carga útil deve receber o valor 0. Deve-se retornar o endereço do pacote recém criado;

- Escrita de dados em pacote (*writeData*): Deve-se receber como parâmetro um pacote já criado, um array de bytes contendo a informação de carga útil a ser colocada no pacote e o tamanho do array passado como parâmetro. A função deve obter os dados do array de bytes passados como parâmetro sequencialmente, mas deve respeitar o limite da carga útil do pacote. Caso o array passado como parâmetro seja menor que o tamanho da carga útil do pacote, deve-se colocar o valor 0 nas posições restantes da carga útil;

- Copiar pacote (*clonePacket*): essa função deve receber como parâmetro um pacote de rede e deve retornar o endereço de memória de uma cópia do pacote passado como parâmetro. Após a cópia, os dois pacotes terão o mesmo conteúdo, mas devem estar colocados em espaços distintos de memória;

- Imprimir dados do pacote (printPacketInformation): essa função deve receber um pacote como parâmetro e imprimir todos as suas informações, no formato especificado nos exemplos a seguir:

```
1        Packet type: ACK
2        ID: 1
3        Source Address: 10
4        Destination Address: 20
```

```
1        Packet type: DATA
2        ID: 5
3        Source Address: 50
4        Destination Address: 80
5        Payload Size: 10
6        Payload: 33 11 12 41 43 55 67 34 4 1
```

- Desalocar pacote (deletePacket): essa função deve receber um pacote como parâmetro (por referência) e desalocar a memória utilizada para armazenar este pacote. Note que antes de desalocar um pacote (passando o ponteiro do pacote como parâmetro para a função *free()*), deve-se desalocar os atributos que foram alocados dinâmicamenete (no caso, a carga útil do pacote). Após desalocar os atributos alocados dinamicamente, pode-se desalocar o pacote.