1.1) Apontar para o endereço de memória de uma variável/array, armazenando seu endereço. Dessa forma é possível fazer alterações na variável original utilizando apenas o ponteiro que guarda seu endereço.

1.2)

```
struct No {
   int val;
   struct No *prox;
}
typedef struct No No;

No *criaNovo(int val) {
   No *novo = (No*)malloc(sizeof(No));
   if(novo == NULL) {
      printf("Erro!");
   } else {
      novo->val = val;
      novo->prox = NULL;
   }
   return novo; // o elemento retornado será um ponteiro do tipo No
}
```

1.3) Sim, é possível ter mais de um ponteiro guardando o mesmo endereço de memória. Dessa forma, a variável pode ser manipulada por qualquer ponteiro que armazene seu endereço. Ex:

```
   int *p1,*p2,num;
   num = 10;
   p1 = &num;
   p2 = &num;
   printf("%d", *p1); -> 10
   printf("%d", *p2); -> 10
```

2.1)

```
struct Carro{
   int numCarro; // ex: 1° carro, 2° carro...
   char modelo[50];
   char cor[15];
   char placa[10];
   struct Carro *prox; //ponteiro indicando o próximo carro
}
```

Como não é possível saber a quantidade exata de carros que entram na UFAM aos sábados de manhã, é aconselhável utilizar uma lista encadeada, assim, as placas serão armazenadas de acordo com a quantidade de carros que entrarem.

2.2) Como em listas encadeadas o programador é responsável pelo gerenciamento de memória, é muito importante: 1° - alocar espaço na memória para armazenar um novo elemento; 2° - verificar se o espaço foi alocado corretamente; 3° - liberar a memória sempre que remover algum elemento, pois caso contrário, esse elemento estará ocupando um espaço desnecessário na memória e não será mais possível acessá-lo.

3.1)

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

typedef struct No {
   char hora[10];
   struct No *prox;
}No;

typedef struct Lista {
   int qtd;
   No *inicio;
}Lista;

void insereFim(No *lista, No *elem) {
   if(lista->prox == NULL) { //caso de parada
      lista->prox = elem;
      elem->prox = NULL;
   } else { //caso geral
      insereFim(lista->prox, elem);
   }
}

No* criaNovo(char hora[]) {
   No *novo = (No*) malloc(sizeof(No));
   if (novo == NULL) {
      printf("Erro ao alocar memória!");
   } else {
      strcpy(novo->hora, hora);
      novo->prox = NULL;
   }
   return novo;
}

void exibe(No *lista) {
   if(lista->prox == NULL) { //caso de parada
```

```
        printf("%s; ",lista->hora);
    } else { //caso geral
        printf("%s; ",lista->hora);
        exibe(lista->prox);
    }
}

int main() {
    int op;
    char hora[10];
    No *elem;
    Lista *cab = (Lista*) malloc(sizeof(Lista));
    cab->inicio = NULL;
    printf("Informe a hora: ");
    scanf("%s", hora);
    elem = criaNovo(hora);
    cab->inicio = elem;
    cab->qtd = 1;
    do{
        printf("0 - Finalizar\n1 - Inserir\n");
        scanf("%d", &op);
        switch(op) {
            case 1:
                printf("Informe a hora: ");
                scanf("%s", hora);
                elem = criaNovo(hora);
                insereFim(cab->inicio, elem);
                cab->qtd = cab->qtd+1;
                break;
            case 0:
                break;
            default:
                printf("Opção inválida!\n");
        }
    } while (op!=0);
    printf("\nTotal: %d\n", cab->qtd);
    printf("Horários: ");
    exibe(cab->inicio);
    return 0;
}
```

3.2)

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

```
typedef struct Distancia {
   float d;
   struct Distancia *prox;
}Distancia;

Distancia* criaNovo(float d) {
   Distancia *novo = (Distancia*) malloc(sizeof(Distancia));
   if (novo == NULL) {
      printf("Erro ao alocar memória!");
   } else {
      novo->d = d;
      novo->prox = NULL;
   }
   return novo;
}

void insereFim(Distancia *lista, Distancia *elem) {
   if(lista->prox == NULL) { //caso de parada
      lista->prox = elem;
      elem->prox = NULL;
   } else { //caso geral
      insereFim(lista->prox, elem);
   }
}

void exibe(Distancia *lista) {
   if(lista->prox == NULL) { //caso de parada
      printf("%.2f ",lista->d);
   } else { //caso geral
      printf("%.2f ",lista->d);
      exibe(lista->prox);
   }
}

int main() {
   int op;
   float ki, kf, d;
   Distancia *elem;
   Distancia *cab = (Distancia*) malloc(sizeof(Distancia));
   cab->prox = NULL;
   printf("Informe as quilometragens inicial e final: ");
   scanf("%f%f", &ki, &kf);
   d = kf - ki;
```

```
    elem = criaNovo(d);
    cab->prox = elem;
    do{
        printf("0 - Finalizar\n1 - Inserir\n");
        scanf("%d", &op);
        switch(op) {
            case 1:
                printf("Informe as quilometragens inicial e final: ");
                scanf("%f%f", &ki, &kf);
                d = kf - ki;
                elem = criaNovo(d);
                insereFim(cab->prox, elem);
                break;
            case 0:
                break;
            default:
                printf("Opção inválida!\n");
        }
    } while (op!=0);
    printf("Distâncias percorridas: ");
    exibe(cab->prox);
    return 0;
}
```