**Curso Tecnólogo em Sistemas para Internet**



**Disciplina de Programação Estruturada**

**Avaliação**

Nome: Everton Oliveira Fernandes

Turma: TSI-2AN Data: 01/06/2016

1. Sobre ponteiros, responda:
   1. O que são? Cite um exemplo de sua utilidade.

**Ponteiros são váriaveis especiais que apontam para o endereço de memória de uma outra variavel tendo como utilidade manupilá-las indiretamente.**

* 1. Explique os operadores de ponteiros & e \*.

**&: É quando o ponteiro recebe o endereço de memória de uma variável, vetor ou matriz**

**\*: É quando o ponteiro acessa o conteúdo de uma variável, vetor ou matriz**

* 1. Mostre como atribuir valores para uma variável e para um vetor através de um ponteiro.

|  |  |
| --- | --- |
| **Atribuição a uma váriavel:**  **main(){**  **int \*p,x=10;**  **p=&x;**  **(\*p)++;**  **printf("%i ",\*p);**  **}** | **Atribuição a um vetor:**  **main(){**  **int vet[4],\*p,i;**  **p=vet;**  **for(i=0; i<4; i++){**  **\*(p+i)=i;**  **printf("vet[%i]=%i\n",i,vet[i]);**  **}**  **}** |

* 1. Considerando p um ponteiro, explique a diferença entre entre os comandos (\*p)++ e p++.

**p++; É quanto o ponteiro avança para a próxima posição de um vetor por exemplo.**

**p=vet;**

**p++;**

**agora o p aponta para &vet[1];**

**Ou melhor dizendo ele recebe o outro endereço de memória, e pode perder a referência da variavel antiga.**

**(\*p)++; É quando através do ponteiro acessamos o conteúdo da variável apontada e incrementamos o conteúdo desta váriavel, ex:**

**Int x=10, \*p;**

**p=&x;**

**(\*p)++; /// x era 10, e agora vira 11.**

**printf(“%i”,\*p);**

* 1. Qual a diferença entre ponteiros do tipo char e int?

**Ambos apontam para o endereço de memória de uma váriavel, a diferença é que ponteiros int apontam somente para variaveis int e ponteiros char apontam somente para variaveis do tipo char. Melhor dizendo, a diferença é o tamanho em bytes, int é 4 bytes vezes o tamanho do vetor, e char é 1 byte vezes o tamanho do vetor**

1. Sobre funções, responda:
   1. O que são? Cite pelo menos três benefícios de sua utilização.

**Funções são trechos de código que realizam uma determinada tarefa dentro de um programa no qual tem como beneficios: evitar repetições dentro de um programa, reaproveitamento de código, organização, manutenção e otimização do mesmo.**

* 1. O que são passagem de parâmetros para funções? Explique como funciona os dois tipos. Mostre um exemplo de cada.

**Temos dois tipos, por valor e por referência. Por valor é quando enviamos uma cópia das variaveis a função, e por referência é quando enviamos o endereço de memória da variavel para a função assim manipulando as variaveis indiretamente, pois na função quem recebe os endereços de memória das variáveis são variaveis ponteiros.**

***Exemplos:***

|  |  |
| --- | --- |
| **Passagem de parametros por valor:**  **void soma(int a, int b){**  **int resultado=a+b;**  **printf("%i ",resultado);**  **}**  **main(){**  **int n1=5,n2=15;**  **soma(n1,n2);**  **}** | **Passagem de por referência:**  **void incrementa(int \*p1){**  **(\*p1)++;**  **}**  **main(){**  **int n1=10;**  **soma(&n1);**  **printf("%i ",n1);**  **}** |

* 1. Para que serve o comando return? Explique como funciona o retorno das funções.

**O comando return é uma comando reservada do sistema que serve para retornar a função principal a resposta de um determinado tipo de função, no qual seu funcionamento é da seguinte forma: Os valores são passados por parâmetros para as funções de tipo int, char, float ou double, pois somente funções desse tipo possuem retorno, estes valores são manipulados nessa função, podendo ser feita como por exemplo uma soma, uma multiplicação delas e após este procedimento ser realizado atráves do comando *return* retornamos a função main a resposta dessa operação.**

* 1. Mostre um exemplo de uma função void, int e char, bem como suas respectivas chamadas.

|  |  |  |
| --- | --- | --- |
| void hello\_word(){  printf("Hello Word");  }  main(){  hello\_word();  } | int multiplicacao(int \*p1, int \*p2){  int resultado=\*p1\*(\*p2);  return resultado;  }  main(){  int n1=5,n2=5;  int r=multiplicacao(&n1,&n2);  printf("%i ",r);  } | char retorna\_maiuscula(char l){  if(l>=97 && l<=122){  return l-=32;  }  }  main(){  printf("Minuscula: ");  char letra=getche();  char maiuscula=retorna\_maiuscula(letra);  printf("\nMaiuscula: %c",maiuscula);  } |

1. Analise o trecho de código abaixo:

int aleatorio(int inf, int sup){ srand(time(NULL));

return rand()%(sup-inf+1)+inf;

}

main(){

int a,b;

scanf("%i %i",&a,&b);

printf("%i",aleatorio(a,b));

}

Reescreva o trecho de código, transformando a função *aleatório* em um procedimento (função sem retorno), e removendo a passagem de parâmetros, fazendo com que a comunicação entre as funções *main* e *aleatorio* se dê exclusivamente pelo uso de variáveis globais.

1. Analise o trecho de código abaixo, explicando o que acontece em cada linha, e diga o que aparece na tela ao fim da execução:

int f[5]={1,2,3,4,5},\*p,\*q,i,j; p=f; q=f+4;

for (i=0 ; i<5 ; i++)

\*(p+i)=\*(q-i);

\*p = \*(q-2);

\*(q-\*p)-=\*p+\*q;

q--; p++;

\*p+=\*q;

f[3] = 2+\*q\*(\*p);

for (p=f ; p<=q; p++) printf("%i",\*p);

1. Sobre alocação dinâmica, responda:
   1. O que é, e para que serve?

**É quando alocamos espaços de memória durante a execução do programa, aumentando ou diminuindo a quantidade de memória alocada, assim não desperdiçando recursos e alocando somente o que é necessário não sobrecarregando a máquina.**

* 1. Crie um trecho de código que aloque um espaço para 5 números inteiros, e preencha-os com valores informadores pelo usuário. Desaloque o espaço alocado ao fim.

|  |
| --- |
| **main(){**  **int tam=5,i;**  **int \*v=(int\*)malloc(sizeof(int)\*tam); ///espaço alocado para 5 inteiros (20 bytes)**  **for (i=0; i<tam; i++){**  **scanf("%i",&v[i]); ///preenchendo o vetor dinâmico**  **}**  **for (i=0; i<tam; i++) {**  **printf("%i ",v[i]);**  **}**  **free(v); /// Espaço desalocado ao fim do programa**  **}** |

1. Observe o trecho de código abaixo:

int remove(int vet[], int t){ t--;

int r = vet[t];

vet = (int\*)realloc(vet,sizeof(int)\*t); return r;

}

Supondo que a função *remove* deve receber um conjunto de números *vet* de tamanho *t*, retornar o último elemento deste conjunto e reduzir em 1 o espaço alocado para estes números (afinal, a função remove um elemento), aponte os erros da função, corrigindo-os, bem como apresente como seria a chamada desta função. A função não apresenta erros de lógica, somente no uso de ponteiros.

|  |
| --- |
| RESPOSTA  int remover(int \*\*vet, int \*t){  (\*t)--;  int r=\*(\*vet+\*t);  \*vet = (int\*)realloc(vet,sizeof(int)\*(\*t));  return r;  }  main(){  int tam=5,i;  int \*vetor=(int\*)malloc(sizeof(int)\*tam);  for(i=0; i<tam; i++){  scanf("%i",&vetor[i]);  }  int ultimo\_elemento=remover(&vetor,&tam);  printf("%i ",ultimo\_elemento);  } |

1. Observe o código abaixo:

int maimen(int v[t]){ int i,m,n;

for (i=0 ; i<t ; i++){

if (v[i] > m || i==0) m = v[i];

if (v[i] < m || i==0) n = v[i];

}

return m,n;

}

Considerando que a função deva receber um conjunto numérico *v* de tamanho *t*, e verificar e atribuir por referência o maior e menor elemento deste conjunto, aponte os erros no código e corrija o mesmo.