

Bruno Rafael de Campos: 13.01737-3

Caio Corrêa: 13.01396-3

Keneth Kendji Yamada: 13.00610-0

Luiz Gustavo Venarusso: 13.02512-0

1) a)

- C: Um compilador é um programa de computador que traduz um programa em linguagem textual para uma linguagem de máquina, específica por um processador e sistema operacional, no caso, faz isso para a linguagem C.
- Assembler: O montador é o programa que transforma o código escrito na linguagem Assembly em linguagem de máquina. O assembler irá substituir as instruções e variáveis para códigos binários e endereços de memórias correspondentes para máquina.
- Linker: é um programa que liga os objetos, na qual um compilador se encarregada de compilar, deixando as referências mais abstratas para as mais concretas. O ligador recebe como entrada um conjunto de arquivos objeto, bibliotecas e parâmetros na linha de controle e produz como resultado um arquivo objeto de saída.

b) Um Sistema Operativo em Tempo Real é um sistema operacional à execução de várias tarefas onde se defina com antecedência o tempo de resposta a um evento para funções com alto grau de precisão e confiabilidade, caso não seja possível o término da tarefa, esta é dada como uma falha no sistema. Uma utilização bem importante com esse sistema é o de airbag que necessita de precisão e confiabilidade, e o estímulo deve ser feito em uma determinada fração de tempo, característica típica de um RTOS.

c) Modelo em V é um modelo conceitual que visa a melhoria ao problema de reatividade do modelo em cascata. O modelo permite que os testes sejam feitos contra os próprios requisitos do componente testado em questão, enquanto os modelos anteriores testavam a especificação. As principais características são:

- Maior efetividade nos testes;
- Este modelo possibilita encontrar erros;
- Ajuda a desenvolver novos requisitos;
- Melhora a qualidade do produto resultante, validando o processo.

d) C++ é uma extensão do C, mas essas linguagens possuem suas especificações:

- O C permite a conversão implícita entre o tipo de dado void\* para ponteiros, algo que enquanto o C++ não permite.
- O C permite que constantes de caracteres sejam inseridas em chamadas de funções com parâmetros tipo char\*, já em C++ é preciso declarar o parâmetro const char\*;
- A diferença na montagem de código entre as duas linguagens as tornam fundamentalmente diferentes.
- C é orientado a procedimentos, já o C++ é voltado a orientação a objetos.

- C++ permite a programação em modo misto, pode se escrever partes do código orientadas a procedimentos e outras orientadas a objetos.

2)

2.1)

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

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

```
int main()
```

```
{
```

```
    printf("Eletronica Embarcada \n Aula 2 - Revisao C \n SeuNome, data\n");
```

```
}
```

2.2)

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

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

```
int main(void)
```

```
{
```

```
    int i;
```

```
    for (i=0;i<101;i++)
```

```
    {
```

```
        printf("%d; ",i);
```

```
    }
```

```
}
```

2.3)

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

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

```
int main(void)
```

```
{
```

```

int vetor[20]= {2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71};

int i,j;

i = 0;

    for(j=0; j<20; j++)
    {
        i++;
        printf("Primo %d: %d\n",i,vetor[j]);
    }
}

```

2.4)

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

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

```

int maximodc(int a, int b)
{
    int i,mdc;
    for(i=1; i<=a || i<=b; i++)
    {
        if(a%i==0 && b%i==0)
            mdc=i;
    }
    return mdc;
}

```

```

int main(void)
{
    int num1,num2,mdc;

    printf("Digite o numero1: \n");
    scanf("%d",&num1);

```

```

printf("Digite o numero2: \n");

scanf("%d",&num2);

maximodc(num1,num2);

printf("O MDC de %d e %d: %d",num1,num2,maximodc(num1,num2));
}

```

```

C:\Users\Keneth\Desktop\jdi\bin\Debug\jdi.exe
Digite o numero1:
20
Digite o numero2:
10
O MDC de 20 e 10: 10
Process returned 20 (0x14)   execution time : 5.694 s
Press any key to continue.

```

2.5) A prototipagem é uma declaração que omite o corpo, mas que especifica os argumentos da função. Tem a utilidade de especificar a interface, declarando uma função, facilitando a leitura do programa e deixando-o mais organizado. Após as declarações das funções, o programa main vem logo em seguida e, somente depois, a programação das funções são feitas.

2.6)

Count = 20

\*temp = 20

Sum = 20;

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      int count = 10, *temp, sum = 0;
7
8      temp = &count;
9      printf("count = %d, *temp = %d, sum = %d\n", count, *temp, sum);
10     *temp = 20;
11     printf("count = %d, *temp = %d, sum = %d\n", count, *temp, sum);
12     temp = &sum;
13     printf("count = %d, *temp = %d, sum = %d\n", count, *temp, sum);
14     *temp = count;
15     printf("count = %d, *temp = %d, sum = %d\n", count, *temp, sum);
16 }

```

```

C:\Users\Keneth\Desktop\teste\bin\Debug\teste.exe
count = 10, *temp = 10, sum = 0
count = 20, *temp = 20, sum = 0
count = 20, *temp = 0, sum = 0
count = 20, *temp = 20, sum = 20
Process returned 33 (0x21)   execution time : 0.038 s
Press any key to continue.

```

2.7)

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

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

```
void swap (int *p1, int *p2){
```

```
    int temp ;
```

```
    temp = *p1;
```

```
    *p1=*p2;
```

```
    *p2= temp;
```

```
}
```

```
int main()
```

```
{
```

```
    int x=10,y=20;
```

```
    swap(&x,&y);
```

```
    printf("x: %d, y: %d\n", x, y);
```

```
}
```

2.8)

A frase anterior é falsa já que o nome de um vetor já é um ponteiro para o primeiro elemento desse mesmo vetor, que pode ser acessado também por \*vetor.

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

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

```
int main(void){
```

```
    int v[6] ={1,3,5,7,9,115};
```

```
    printf("O acesso funcionou!: %d\n",v[5]);
```

```
    printf("O acesso funcionou de novo!: %d",*(v+5));
```

```
}
```

```
main.c x
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(void){
5      int v[6] = {1,3,5,7,9,115};
6      printf("0 acesso funcionou!: %d\n",v[5]);
7      printf("0 acesso funcionou de novo!: %d",*(v+5));
8  }
9
```

```
C:\Users\Keneth\Desktop\teste\bin\Debug\teste.exe
0 acesso funcionou!: 115
0 acesso funcionou de novo!: 115
Process returned 32 (0x20)   execution time : 0.040 s
Press any key to continue.
```

2.9)

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

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

```
void print_array(int *a){
```

```
    int i;
```

```
    for(i=0;i<20;i++){
```

```
        printf("%d; ",a[i]);
```

```
    }
```

```
}
```

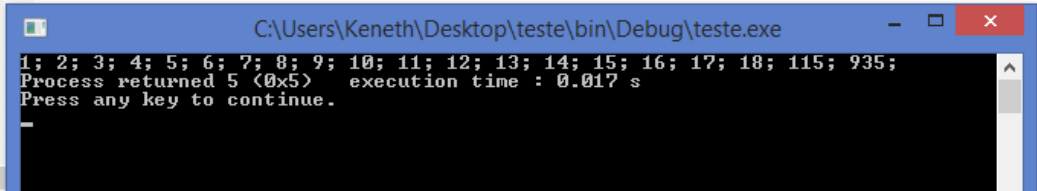
```
int main(void){
```

```
    int test_array[20] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,115,935};
```

```
    print_array(test_array);
```

```
}
```

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void print_array(int *a){
5      int i;
6      for(i=0;i<20;i++){
7          printf("%d; ",a[i]);
8      }
9  }
10
11 int main(void){
12     int test_array[20] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,115,935};
13     print_array(test_array);
14 }
15
```



O parâmetro correto é uma variável que esteja ligada a um ponteiro para que a varredura seja feita de forma correta no vetor.

2.10)

O erro está no return v que só retorna o endereço do vetor e não mostra os valores atribuídos em cada endereço do vetor.

2.11)

A função malloc aloca um bloco de bytes consecutivos na memória e retorna o endereço do bloco armazenado. O endereço devolvido por malloc é do tipo genérico void, esse endereço é armazenado em um ponteiro. Após a utilização do malloc deve-se utilizar a função free, que libera o espaço de memória que foi utilizado no momento da alocação.