Exame Recurso 2018/2019

1. Considere uma lista de inteiros L, como definida nas aulas. O seguinte código permite:

```
NO *p = L->inicio;
while (p)
{
  printf("Info= %d\n",p->info);
  p = p->prox;
};
```

- [x] Mostrar todos os elementos da lista;
- [ ] Mostra somente o primeiro elemento da lista;
- [ ] Entra em ciclo infinito;
- [ ] Nenhuma das anteriores ou existem erros na execução.

2. Considere uma lista de inteiros L, como definida nas aulas. O seguinte código:

```
NO *p = L->inicio;
while (p)
{
  p = p->prox->prox;
  if (p) p->info = 0;
};
```

- [ ] Incrementa uma unidade a todos os elementos da lista;
- [ ] Coloca a zero os elementos da lista a partir do segundo elemento;
- [ ] Vai causar um erro aquando da execução;
- [x] Nenhuma das anteriores ou existem erros de compilação.

3. Considere uma árvore binária ordenada (crescente), onde foram inseridos os valores (por esta ordem) 50; 15; 30; 10; -5 e o seguinte código:

```
int F(NO *p) {
if (!p) return 0;
return 1+Minimo(F(p->Esq),F(p->Dir));
}
```

A seguinte chamada:

```
printf("Valor = %d", NOa(K->raiz));
```

- [ ] Tem como output 4;
- [ ] Tem como output 3;
- [ ] Tem como output 1;
- [x] Nenhuma das anteriores ou tem erros de sintaxe.

se tivesse F seria 1

4. Considere uma árvore binária K ordenada (crescente), onde foram inseridos os valores (por esta ordem) 50; 20; -5; 5; 6; 30; 10.

```
int Pert(NO *p, int X)
{ if (!p) return 0;
  if (p->info == X) return 1;
  if (X>p->info) return Pert(p->Esq,X);
  if (X<p->info) return Pert(p->Dir,X);
}
```

A seguinte chamada:

```
printf("Valor = %d",Pert(K->raiz, 15));
```

- [ ] Tem como output 1;
- [x] Tem como output 0;
- [ ] Entra em ciclo infinito;
- [ ] Nenhuma das anteriores ou existem erros de sintaxe.

5. Considere o seguinte código (a estrutura Livro tem os campos **codigo**(int) e **preco**(float)):

```
Livro VP;
VP = (Livro *)malloc(20*sizeof(Livro));
```

Pretende-se acrescentar o iva (23%) a cada livro, qual a instrução correcta?

- [ ] for (i=0; i<20; i++) VP[i]->preco+=23/100;
- [ ] for (i=0; i<20; i++) VP[i]->preco *=1.23;
- [x] for (i=0; i<20; i++) VP[i].preco *= 1.23;
- [ ] Nenhuma das anteriores ou existem erros de sintaxe.

6. O conjunto de instruções:

```
int *p, x;
p = &x;
x = 5;
while (*p)
 printf("Valor de p = %d\n", (*p)--);
```

- [ ] Escreve todos os números de 5 até 0;
- [x] Escreve todos os números de 5 até 1;
- [ ] Entra em ciclo infinito;
- [ ] Nenhuma das anteriores ou existem erros de sintaxe.

7. O código:

```
int *Q = (int *)malloc(sizeof(int));
int *L = (int *)malloc(sizeof(int));;
*Q = 10;
*L = 20;
*L = *Q;
*Q = *L;
printf("L = [%d]; Q = [%d];\n", *L, *Q);
free (L); free (P);
```

Qual o resultado?

Q

- [ ] L =[20]; Q = [10];
- [ ] L =[10]; Q = [20];
- [x] L =[10]; Q = [10];
- [ ] Nenhuma das anteriores.

8. Considere uma lista de Pixeis, Assuma que existe uma função DestruirLista; que vai destruir a lista e todo o seu conteudo.

```
Lista *L1 = (Lista *)malloc(sizeof(Lista));
Lista *L2 = (Lista *)malloc(sizeof(Lista));
PIXEL *P = (PIXEL *)malloc(sizeof(PIXEL));
Add(L1, P);
Add(L2, P);
DestruirLista(L1); DestruirLista(L2);
```

- [ ] cria duas Listas e cria dois PIXEL;
- [ ] cria duas Listas e cria dois PIXEL, destruindo de seguida as listas;
- [ ] As instruções têm erro de compilação;
- [x] Podem acontecer resultados inesperados na execução do último DestruirLista.

Exame Normal(?) 2018/2019

1. Considere uma lista de inteiros L, como definida nas aulas. O seguinte código permite:

```
NO *p = L->inicio;
while (p)
{
  p = p->prox;
  printf("Info= %d\n",p->info);
};
```

- ☐ Mostrar todos os elementos da lista;
- ☐ Mostra somente o primeiro elemento da lista;
- ☐ Entra em ciclo infinito;
- ☒ Nenhuma das anteriores ou existem erros na execução.

2. Considere uma lista de inteiros L, como definida nas aulas. O seguinte código:

```
NO *p = L->inicio;
while (p)
{
  if (p) p->info = 0;
  p = p->prox;
};
```

- ☐ Incrementa uma unidade a todos os elementos da lista;
- ☐ Coloca a zero os elementos da lista a partir do segundo elemento;
- ☐ Vai causar um erro aquando da execução;
- ☒ Nenhuma das anteriores ou existem erros de compilação.

3. Considere uma árvore binária ordenada (crescente), onde foram inseridos os valores (por esta ordem) 50; 15; 30; 10; -5 e o seguinte código:

```
int NOa(NO *p) {
if (!p) return 1;
  return  1+Maximo(NOa (p->Esq),
                        NOa(p->Dir));
}
```

A seguinte chamada:

```
printf("Valor = %d", NOa(K->raiz->esq));
```

- ☒ Tem como output 4;
- ☐ Tem como output 3;
- ☐ Entra em ciclo infinito;
- ☐ Nenhuma das anteriores ou tem erros de sintaxe.

4. Considere uma árvore binária K ordenada (crescente), onde foram inseridos os valores (por esta ordem) 50; 20; -5; 5; 6; 30; 10.

```
int Pert(NO *p, int X)
{ if (!p) return 0;
  if (p->info != X) return 1;
  if (X>p->info) return Pert(p->Dir,X);
  if (X<p->info) return Pert(p->Esq,X);
}
```

A seguinte chamada:

```
printf("Valor = %d",Pert(K->raiz, 15));
```

- ☒ Tem como output 1;
- ☐ Tem como output 0;
- ☐ Entra em ciclo infinito;
- ☐ Nenhuma das anteriores ou existem erros de sintaxe.

5. Considere o seguinte código (a estrutura Livro tem os campos **codigo**(int) e **preco**(float)):

```
Livro *VP;
VP = (Livro *)malloc(20*sizeof(Livro));
```

Pretende-se acrescentar o iva (23%) a cada livro, qual a instrução correcta?

- ☐ for (i=0; i<20; i++) VP[i]->preco+=23/100;
- ☒ for (i=0; i<20; i++) VP[i]->preco *=1.23;
- ☐ for (i=0; i<20; i++) VP[i].preco *= 1.23;
- ☐ Nenhuma das anteriores ou existem erros de sintaxe.qd temos * usamos '->' ; outros casos usamos '.'

6. O conjunto de instruções:

```
  int *p, x;
  p = &x;
  x = -11;
  while (*p)
   printf("Valor de p = %d\n", (*p)--);
```

- ☐ Escreve todos os números de 10 até 0;
- ☐ Escreve todos os números de 11 até 1;
- ☒ Entra em ciclo infinito;
- ☐ Nenhuma das anteriores ou existem erros de sintaxe.

7. O código:

```
int *P = (int *)malloc(10*sizeof(int));
for (int i = 0; i < 10; i++)
     P[i] = 10*i;
int *Q = P + 8;P[8]=80
int *L = Q - 2;P[8]-2=P[6]=60
```

Qual o resultado de *Q - *L ?80-60=20

- ☐ 30;
- ☒ 20;
- ☐ 0
- ☐ Nenhuma das anteriores.

8. O código:

```
Caixa *M = (Caixa *)malloc(sizeof(Caixa));
free (M);
```

- ☒ cria um novo ponteiro M alocando espaço para uma caixa e depois liberta-o;
- ☐ cria um novo ponteiro M para uma caixa e depois liberta-o;
- ☐ A(s) instruções têm erro de compilação;
- ☐ Nenhuma das anteriores.

Exame Recurso 2016/2017

1. Considere uma lista de inteiros L, como definida nas aulas. O seguinte código permite:

```
NO *p;
while (p)
{
  printf("Info= %d\n",p->info);
  p = p->prox;
};
```

- [x] Mostrar todos os elementos da lista;
- [ ] Mostra somente o primeiro elemento da lista;
- [ ] Entra em ciclo infinito;
- [ ] Nenhuma das anteriores ou existem erros na execução.

2. Considere uma lista de inteiros L, como definida nas aulas. O seguinte código:

```
NO *p = L->inicio;
while (p)
{
  p = p->prox;
  if (p) p->info = 0;
};
```

- [ ] Incrementa uma unidade a todos os elementos da lista;
- [x] Coloca a zero os elementos da lista a partir do segundo elemento;
- [ ] Vai causar um erro aquando da execução;
- [ ] Nenhuma das anteriores ou existem erros de compilação.

3. Considere uma árvore binária K ordenada (inicialmente vazia) e o seguinte código:

```
int NOa(NO *p) {
 if (!p)
   return 0;
 return 1+Maximo(NOa(p->esq),
                 NOa(p->dir));
}
```

Após inserir os elementos -50; -15; -30; -10; -5 (por esta ordem), a seguinte instrução:

```
printf("Valor = %d", NOa(K->raiz->esq));
```

- [ ] Tem como output 4;
- [x] Tem como output 0;
- [ ] Entra em ciclo infinito;
- [ ] Nenhuma das anteriores ou tem erros de sintaxe;

4. Considere uma árvore binária K ordenada (crescente), onde foram inseridos os valores (por esta ordem) 2; 3; 4; 5; 6;

```
int Fc(NO *p)
{
 if (!p)
   return 0;
 return p->info*Fc(p->Dir)*Fc(p->Esq);
}
```

A seguinte chamada:

```
printf("Valor = %d",Fc(K->raiz));
```

- [ ] Tem como output 1;
- [x] Tem como output 0;
- [ ] Tem como output 720;
- [ ] Nenhuma das anteriores ou existem erros de sintaxe;

5. Considere o seguinte código (a estrutura Livro tem os campos **codigo**(int) e **preco**(float)):

```
Livro *VP;
VP = (Livro *)malloc(N * sizeof(Livro));
```

Pretende-se implementar uma promoção, descendo 10% o preço de cada livro. Qual o código correcto para o conseguir?

- [ ] for (i=0; i<N; i++) VP[i]->preco-=10/100;
- [ ] for (i=0; i<N; i++) VP[i]->preco /=1.1;
- [ ] for (i=0; i<N; i++) VP[i].preco *= 0.9;
- [x] Nenhuma das anteriores ou existem erros de sintaxe;

6. O conjunto de instruções:

```
int *p, x;
p = &x;
x = -11;
while (*p)
 printf("Valor de p = %d\n", (*p)--);
```

- [ ] Escreve todos os números de 10 até 0;
- [ ] Escreve todos os números de 11 até 1;
- [x] Entra em ciclo infinito;
- [ ] Nenhuma das anteriores ou existem erros de sintaxe;

7. Considere o seguinte código:

```
int *P = (int *)malloc(10*sizeof(int));
for (int i = 0; i < 10; i++)
     P[i] = 10*i;
int *Q = P + 8;
int *L = Q - 3;
```

P[8]=80

P[8]-3=P[5]=50

Qual o resultado de *Q - *L ?

80-50=30

- [x] 30;
- [ ] 20;
- [ ] 0;
- [ ] Nenhuma das anteriores;

8. Um restaurante pretende implementar um sistema na cozinha que gerir os pedidos (pratos) dos seus clientes. Qual a estrutura que mais de adequa?

- [ ] Lista; adicionas a um sitio qlq
- [x] Fila; o primeiro a entrar]e o primeiro a sair
- [ ] Pilha; tiramos sempre o ultimo
- [ ] Árvore Binária;

Árvores binárias

1)contar número de folhas da árvore (contarFolhas)

```
int conta(ELEMENTO *A)
{
        if(!A) return 0;
        if (!A->esq && !A->dir) return 1;
        int c_folhas_esq = conta(A->esq);
        int c_folhas_dir = conta(A->dir);
        return c_folhas_esq + c_folhas_dir;
}

int contarFolhas(ABinaria *A)
{
        if(!A) return 0;
        int folhas = conta(A->raiz);
        return folhas;
}
```

2)eliminar todas as folhas da arvore(destruirfolhasarvore) (feita por mim, pode estar mal)

```
void DestruirFolhas(NO_ARV *P)
{
if (P == NULL) return;
if (!A->esq && !A->dir)
   {
    DestruirRamosFolhas(P->Esq);
    DestruirRamosFolhas(P->Dir);
   }
free(P->Info);
free(P);
}

void DestruirFolhasArvore(Arv_Binaria *AB)
{
 if (AB)
     {
      DestruirRamosFolhas(AB->Inicio);
      free(AB);
      }
}
```

3)contar o numero de nós que não são folhas (feita por mim, pode estar mal)

```
Int contarNaoFolhas(ABinaria *A)
{
        if(!A) return 0;
        return naoFolhas(A->raiz);
}
```

```
Int naoFolhas(ELEMENTO *A)
{
        if(!A) return 0;
        if (A->esq || A->dir) return 1;
        return naoFolhas(A->esq) + naoFolhas(A->dir);
}
```

4) contar nós da árvore num intervalo

```
int Contar_Nos_Idade_Intervalo(ARVORE *Arv, int valor_min, int valor_max)
{
if (!Arv) return -1;
return Contar_Nos_Intervalo(Arv->raiz, valor_min, valor_max);
}

int Contar_Nos_Intervalo(NO *P, int valor_min, int valor_max)
{
    if (!P) return 0;
    int Contador = 0;
    for (int i = 0; i < MAX_FILHOS; i++)
        Contador += Contar_Nos_Intervalo(P->V[i], valor_min, valor_max);
    if ((P->inf->idade > valor_min) && (P->inf->idade < valor_max))
        return 1 + Contador;
    return Contador;
}
```

5) colocar todos os elementos da arvore numa ZONA(colocarArvoreZona) MINERD

```
void ColocarArvoreZona(ABinaria *A, ZONA *Z)
{
if(!A || !Z) return;
Z->raiz = (NO_PIXEL*) malloc(sizeof(NO_PIXEL));
Z->raiz->Info=NULL;
Z->raiz->proximo = NULL;
NO_ARVORE *P = A->raiz;
NO_Pixel *L = Z->raiz;
colocarZona(P, Z);
}

NO_Pixel colocarZona (NO_ARVORE *P; NO_Pixel *L ){
if(!P || !L) return;
L->Info = P->info;
if(P->esq)
   {
    L->proximo = (NO_Pixel*) malloc(sizeof(NO_Pixel));
    L = colocarZona(P->esq, L->proximo);
   }
if(P->dir)
   {
    L->proximo = (NO_Pixel*) malloc(sizeof(NO_Pixel));
    L = colocarZona(P->dir, L->proximo);
```

```
    }
return L;
}
```

6) Contar o numero de Pixeis da árvore que estão “próximos”, de um outro pixel (R,G,B) a menos de uma tolerância D. (ContarPixeisSemelhantes) MINERD

```
int ContarPixeisSemelhantes(ABinaria *A, int R, int G, int B, int D)
{
if(!A) return;
int cont = 0, min, max;
NO_ARVORE *P = A->raiz;
min = R+G+B;
min = min/3;
max = min + D;
min = min - D;
cont = CertificarSemelhanca(P, cont, max, min);
return cont;
}

int CertificarSemelhanca(NO_ARVORE *NO, int cont, int max, int min)
{
if(!NO) return cont;
int tot;
tot = NO->Info->R + NO->Info->B + NO->Info->G;
tot = tot/3;
if(tot<max && tot>min) cont++;
CertificarSemelhanca(A->esq);
CertificarSemelhanca(A->dir);
return cont;

}
```

7) Criar Árvore Binária

```
Arv_Binaria *CriarArvBinaria()
{
Arv_Binaria *Ab = (Arv_Binaria *)malloc(sizeof(Arv_Binaria));
if (!Ab) return NULL;
Ab->Inicio = NULL;
Ab->NEL = 0;
return Ab;
}
```

8) Destruir Árvore toda ig

```
void DestruirRamosFolhas(NO_ARV *P
{
if (P == NULL) return;
DestruirRamosFolhas(P->Esq);
DestruirRamosFolhas(P->Dir);
free(P->Info);
```

```
free(P);
}

void DestruirArvBinaria(Arv_Binaria *AB)
{
if (AB)
   {
    DestruirRamosFolhas(AB->Inicio);
    free(AB);
    }
}
```

9) Mostrar arvore binaria

```
void MostrarRamosFolhas(NO_ARV *P)

{
if (!P) return;
MostrarRamosFolhas(P->Esq);
MostrarPessoa(P->Info);
MostrarRamosFolhas(P->Dir);
}

void MostrarArvBinaria(Arv_Binaria *AB)
{
if (!AB) return;
printf("ARV.Binaria NEL = %d\n", AB->NEL);
MostrarRamosFolhas(AB->Inicio);
}

void MostrarPessoa(PESSOA *K)
{
if (K)
printf("PESSOA: CC = %d\n", K->CC);
}
```

10) Altura Árvore

```
int Altura_Nos(NO *P)
{
if(!P) return 0;
int Alt_max=0;
for(int i=0; i<MAX_FILHOS; i++)
{
Alt_aux = Maior(Altura_Nos(P->V[i],Alt_aux);
}
return 1+Alt_aux;
```

```
int Altura(ARVORE *Arv)
{
if (!Arv) return -1;
return Altura_Nos(Arv->raiz);
}
```

Listas

Funcao apoio:

```
NO criarNo (Informacao* pElemento)
{
NO p = (NO)malloc(sizeof(No));
p->inf = pElemento;
p->prox = NULL;
return p;
}
```

Criar Lista

```
Lista* criarLista()
{
LISTA lista; // ou Lista *lista;
lista = (LISTA)malloc(sizeof(Lista));
lista->inicio = NULL;
lista->nElementos = 0;
return lista;
}
```

Criar elemento

```
Informacao *criarUmElemento(char *nome, int num, float media)
{
Informacao *A = (Informacao *)malloc(sizeof(Informacao));
//A fica a apontar para o bloco criado, do tipo Informacao
strcpy(A->nome, nome);
A->numero = num;
A->mediaCurso = media;
return A;
}
```

Gravar elemento ficheiro

```
void gravarElementoFicheiro(FILE * f, Informacao * a)
{
fprintf(f, "Nome-do-Aluno= %s\n", a->nome);
fprintf(f, "Numero-do-Aluno= %d\n", a->numero);
fprintf(f, "Media-do-Curso= %f\n", a->mediaCurso);
}
```

Ler elemento ficheiro

```
Informacao * lerElementoFicheiro(FILE * f)
{
char nome[MAX_NOME];
int num;
float media;
fscanf(f,"%s", nome);
fscanf(f,"%d", &num);
fscanf(f,"%f", &media);
return criarUmElemento(nome, num, media);
}
```

Função iterativa para verificar se um dado Elemento pertence à lista

```
int pertenceIterativo(LISTA l, Informacao * a)
{
if (a == NULL || l == NULL) return 0;
NO p = l->inicio;
    while (p != NULL)
    {
      if (elementosIguais(p->inf, a))
      return 1;
      else
      p = p->prox;
     }
return 0;
}
```

Mostrar todo o conteúdo da lista pela ordem em que se encontram os elementos

```
void listarLista(LISTA l)
{
    if (l == NULL) return;
    NO p = l->inicio;
    while (p != NULL)
    {
        mostrarInformacao (p->inf);
         p = p->prox;
     }
}
```

Mostrar todo o conteúdo da lista pela ordem inversa àquela em que se encontram na lista

```
void listarNoContrario(NO p)
{
     if (p)
     {
          listarNoContrario(p->prox);
          mostrarInformacao(p->inf);
      }
}
```

```
void listarContrario(LISTA l)
{
	if (l == NULL) return;
	listarNoContrario(l->inicio);
}
```