Introducción a la Programación

Ejercicios de exámen.

Asegura tu aprobado con nuestros cursos de cálculo

CEUS es una empresa con mas de 50 años de experiencia en el sector de la educación y la formación lo que la hacen la opción ideal para recibir los cursos que está buscando en multitud de ámbitos.

Si está buscando algun tipo de curso en Cádiz, no dude en contactar con nosotros. Nuestro conocimiento del sector le ayudará a encontrar siempre la mejor opción gracias al asesoramiento que nuestra experiencia puede brindarle.

www.ceusformacion.com

99%
satisfacción

CEUS
ONLINE

Estructuras Repetitivas:

- **Definicion:** permiten repetir una o varias veces la función de evaluación de una determinada condición.

- **Estructura MIENTRAS:** Se repite mientras la condición sea verdadera.

  Mientras <condición> hacer
  	Instrucciones
  Fin_mientras

  Y en lenguaje C:

  While <condición> {
    Instrucciones;
  }

- **Estructura repetir:** La condición se encuentra al final del bucle y se ejecutará hasta que se cumpla esa condición.

  Repetir
  	Instrucciones
  Hasta que <condición>

  Y en lenguaje C:

  Do{
  	Instrucciones;
  } while <condición> ;

- **Estructura desde:** Desde un valor de una variable se irá incrementando, o decrementando, hasta llegar a un valor especificado:

  Desde var1 <- 'valor' hasta var2 hacer
  	<actualización>
  	Instrucciones
  Fin_desde

  Y en lenguaje C:

  For (var1 = 'valor' ; var1 < var2 ; <actualización>) {
  	Instrucciones;
  }

- **Sustituciones:** Podemos sustituir una estructura desde por un repetir por una mientras. Sin embargo, una mientras por una desde sólo si conocemos el número final de iteraciones, y una mientras por un repetir siempre que no altere el resultado final.

Razona la veracidad o la falsedad de las siguientes afirmaciones:

a) **El modificador static se usa para indicar que una variable estática está definida en otro modulo:** FALSO, se usa para modificar la persistencia de las variables.
b) **En el lenguaje C todos los parámetros se pasan siempre por valor:** VERDADERO, se simula el paso por referencia con los punteros.
c) **Una unión es una colección de variables del mismo tipo y distinto tamaño:** FALSO, pueden ser de tamaños iguales.
d) **El contenido asociado al identificador de un vector en su definición es constante:** FALSO, su contenido puede cambiar, pero no su tipo de dato.
e) **Una variable de tipo puntero sólo puede almacenar una dirección de memoria:** VERDADERO, una sola dirección por puntero.
f) **El operador de indirección * permite obtener la dirección de una variable**. FALSO, se obtiene con el operador &.
g) **El operador & sólo puede aplicarse a variables tipo puntero**. FALSO, por ejemplo && se usa como operador lógico, indicando conjunción.
h) **En C no es posible modificar la persistencia de las variables locales:** FALSO, se puede modificar con el modificador Static.
i) **Para escribir en un fichero necesitamos abrirlo y asociarlo a una variable tipo FILE**: VERDADERO, file debe ser tipo puntero, y no debemos olvidar de cerrar después el fichero.
j) **La función realloc permite ampliar la memoria previamente asignada a un puntero indicándole cuanto desea ampliarlo**: VERDADERO, así podemos hacer un mejor uso de la memoria.

Explica la importancia de cada uno de los objetivos de la programación.

- ➔ Corrección: Antes de desarrollar un programa es fundamental expresar el cprrecto funcionamiento del mismo.
- ➔ Claridad: Las descripciones deben ser claras y legibles.
- ➔ Eficiencia: deben crearse programas eficientes que consuman el menor número de recursos posibles.

Deben emplearse una metodología de la programación adecuada sobre el lenguaje que estemos trabajando para que se satisfagan los objetivos anteriormente especificados.

Razona la veracidad o la falsedad de las siguientes afirmaciones:

a) **La ejecución de una función finaliza únicamente con la sentencia return:** FALSO, puede finalizar después de la sentencia return, por ejemplo puede realizar alguna asignación posterior.
b) **Los parámetros que se utilizan en la definición de una función se denominan parámetros formales:** VERDADERO, los que recibe se denominan parámetros actuales.
c) **Las cadenas de caracteres deben definirse al menos con una dimensión igual a la longitud de la cadena más larga que puedan contener**: VERDADERO, si no la cadena más larga a dicha definición no podría almacenarse.
d) **Todas las funciones deben devolver siempre un valor**: FALSO, los procedimientos no devuelven ningún valor.
e) **Cualquien bucle WHILE no puede sustituirse por un FOR en lenguaje C:** FALSO, si podemos siempre que conozcamos el número de iteraciones que se realizará.
f) **Es lo mismo poner int * v, que int v[] en la definición de una función que recibe un vector de enteros:** VERDADERO, ambos se refieren a la primera posición del vector.
g) **Los operadores unarios & y * se utilizan sobre cualquier tipo de variable**: VERDADERO, para trabajar con sus direcciones de memoria.
h) **Las variables globales son aquellas que definimos en el cuerpo de la función main:** FALSO, no sólo en el main, sino en el cuerpo de una función y son de uso exclusivo de esa función.
i) **Extern se usa para definir que una variable local está definida en otra función:** VERDADERO, así declaramos variables externas.
j) **En C el 1 es verdadero, y cualquier valor distinto de 1 es falso:** FALSO, el programador debe especificar dicho valor de verdad.

Indica qué es x en cada una de las siguientes definiciones:

a) **int (*x) (int *, char *) :** Un puntero a función que recibe un puntero a entero, y un puntero a carácter.
b) **int *x[4]:** Un puntero a un vector de entero de longitud 4.
c) **Int (*x)[4]:** Un puntero a entero.
d) **Char x[3][6]:** Matriz de caracteres de 3x6.

Escribe las declaraciones para:

a) **Una función f que recibe como parámetros un puntero a carácter y un puntero a entero y no devuelve ningún valor:** void f (char *p, int *q)
b) **Un puntero a dicha función f:** void *f (char *p, int *q)
c) **Una función fun que recibe como parámetros un puntero a f y un carácter, que devuelve como resultado una cadena de caracteres:** char[x] fun (int (*f), char q)

Realiza las siguientes declaraciones en C, casa una en una línea

a) **V como un vector de 6 cadenas de caracteres de longitud 15:** char v[6][15];
b) **P como una matriz de 3x5 punteros a enteros**: int *p [3][5];
c) **F como un puntero a función que recibe dos parámetros a como una matriz de 3x5 punteros a enteros, y b como un puntero a double, y devuelve un double:** double *f(int a[3][5], doublé *b);
d) **P como un puntero a una matriz de 2x4 punteros a entero:** int *(*p)[2][4];
e) **F1 como una función que recibe un puntero a función (función f2 que devuelve un entero y recibe dos parámetros, a como entero, y b como puntero a entero) y devuelve un carácter:** char f1 (int *f2(int a, int b));

Dado el siguiente fragmento de programa

```
float funcion(int a, floatx[]);
int main() {
int n;
float v[65];
float m;
...
```

¿Cuál de las siguientes llamadas a la función son correctas? Razona la respuesta.

a) **m=funcion(v,n);** Es incorrecta, ya que se está pasando un vector de float como un entero y un entero n como un vector de float.
b) **M=función(n,*v);** Es incorrecta ya que para pasar un puntero usaríamos el operador & en la llamada a la función, aun así en este fragmento de programa no se podría enviar el vector como puntero, puesto que está declarado como float, cosa que es imposible.
c) **M=función(n, v[25]);** Es incorrecta, ya que para pasar un vector no pasamos una determinada longitud, sino un vector en si.
d) **M=función(n,&v);** es correcta.
e) **M=función(n,v)** es incorrecta, ya que habría que usar o &v o v[].

¿A qué hace referencia la persistencia de las variables? Describe brevemente en qué consiste, qué relación existe con el ámbito de las variables, y cómo se pueden clasificar éstas en función de ambos conceptos.

La persistencia de las variables hace referencia al tiempo que pasan dichas variables en memoria.

Como bien sabemos, las variables pueden ser locales (de uso interno en una función) o globales (para su uso en cualquier otra función), que es lo que se conoce como ámbito de las variables.

Pues bien, con el modificador static, podemos modificar la persistencia de dichas variables

Dado el siguiente fragmento de programa, rellena las zonas subrayadas del código, y escribe la cabecera de la función f2

Typedef double vect[100];
Typedef char cad[100];
Typedef struct{
Int a;
Float b;
Vect c;
Cad d;
} T_estructura;
	____ f1 (____a, ____b, float c, ____ d, ____ e) {
	return f2(a->a,e.d,b,&c,d[4]);
	}
	Int main() {
	T_estructura x;
	x.a=f1(&x ,x.d[4], ____, x.c, x);
	}

```
int f1 (T_Estructura a, cad b, float c, vect d, T_estructura e) {
return f2(a->a,e.d,b,&c,d[4]);
}
Int main() {
T_estructura x;
x.a=f1(&x ,x.d[4], x.b, x.c, x);
}
```

La cabecera de la función f2 sería: int f2(int*, cad, char, float*, double);

Dada la definición int v[8]; ¿Qué sentencia, o conjunto de sentencias sería necesario escribir en C si queremos definir v de forma dinámica? ¿Y si después queremos ampliar v para almacenar dos elementos más?

Para definir v de forma dinámica usaríamos: V=(int)malloc(8*sizeof(int));

Y si quisiésemos añadirle dos elementos más: V=(int)realloc(*v,10*sizeof(int));

Dadas las siguientes definiciones en lenguaje C

```
typedef struct {
int codigo;
char nombre[20];
int existencias;
float precio;
} TArticulo;
TArticulo Almacen[100];
TArticulo Producto1, *Producto2;
int indice=1;
char cadena[20];
```

Indica si las instrucciones siguientes son o no correctas razonando la respuesta, indicando en su caso cual sería el error y escribiendo la posible instrucción corregida.

a) **Producto1.nombre=”Martillo”;** FALSO, strcpy(producto1.nombre,”martillo”);
b) **Producto1=almacen[10];** VERDADERO
c) **Printf(“%d”, almacen.existencias);** FALSO, Printf(“%d”, almacen[n].existencias);
d) **Producto1=*Producto2**; FALSO, Producto1=&Producto2;
e) **Índice=Producto2->precio**; FALSO, Indice es tipo int y precio tipo float.
f) **Producto2=&almacen[3];** VERDADERO.
g) **Producto1.código=Producto2->código+Almacen[100].codigo**; FALSO, almacén no puede tomar valores más allá de 99.
h) **Strcpy(Almacen[índice].cadena, “tornillo”);** FALSO, cadena no está definida en TArticulo.
i) **Índice=Producto2.código;** FALSO, Índice=2Producto2->código;
j) **Cadena[2]=Producto1->nombre[índice];** FALSO, Cadena[2]=Producto1.nombre[índice]

Dado el siguiente fragmento de programa:

```
typedef struct{
int a;
char v[20];
char c;
} registro;
Void main() {
        Registro vector[20];
…
```

Relaciona las siguientes cabeceras de funciones correspondientes al programa

void f1(registro *r);

voidf2(registro r);

con sentencias que aparecen a continuación que serían correctas dentro de dichas funciones.

a) **scanf(“%d%c”, &r.a,&r.c);** Correcta para f1.

b) **scanf("%s", r.v);** Correcta para f2.

c) **printf(“%d, %c\n”,r->a, r->c);** Correcta para f1.

d) **printf(%d,%c\n”,(*r).a,(*r).c);** Correcta para f1.

e) **printf(“%c\n”, r[8].v[4]);** Correcta para f2.

f) **printf(“%d, %c\n”,r.a, r.c);** Correcta para f2.

g**) printf(“%c\n”, r->v[5]);** Correcta para f1.

h) **scanf(“%s”, r[3].v);** Correcta para f2.

i) **scanf (“%d%c”,&(*r).a, &(*r).c);** Correcta para f1.

j) **printf(“%d\n”, r[5].a);** Correcta para f2.

Indica la salida del siguiente algoritmo, así como sus parámetros actuales y formales, variables globales y locales, y pasos por valor y por referencia.

ALGORITMO Ejemplo Var entero: a, b, x, y Inicio a←6 b←4 x←5 y←x-3+f(x,a) Escribir(a,b,x,y) Fin algoritmo	Entero función f(S entero:p, E entero, q) Var entero: b Inicio b←x+2*q a←a*2 p←b-a p←(q-p)*2 Escribir(p,b,q) Devolver(p-b+a) Fin funcion

Algoritmo Ejemplo
a←6←12
b←4
x←5←5←2
y←-4

Imprime por pantalla:
2,6,17
12,4,2,-4

Funcion f
b←17
p←5←5←2
q←6

Parámentros por valor: q

Parámetros por referencia: p

Parámetros actuales: x,a

Parámetros formales: p,q

Variables locales: b

Variables globales: a,b,x,y

Traducción a lenguaje C:

```
#include <stdio.h>
#include <stdlib.h>
int a, b, x, y;
int f(int *p, int q) {
int b;
b=x+2*q;
a=a*2;
p=b-a;
p=(q-p)*2;
printf("%d %d %d",p,q,b);
return(p-b+a);
}
```

```
int main() {
a=6;
b=4;
x=5;
y=x-3+f(x,a);
printf("%d %d %d %d", a,b,x,y);
return 0;
}
```

Indica la salida del siguiente algoritmo, así como sus parámetros actuales y formales, variables globales y locales, y pasos por valor y por referencia.

Algoritmo ejemplo Var entero a,b,c,d Inicio a←1 b←2 c←3 d←fun(a,b,c) escribir(a,b,c,d) proc(b*2,c) escribir(a,b,c,d) fin algoritmo	Entero función fun (E entero x, S entero y, S entero z) Var entero: c Inicio y←1 c←y+a+1 z←a+x+c escribe(x,y,z,c) devuelve(c+b) fin funcion	Procedimiento proc (E entero a, E/S entero b) Var entero: z Inicio z←c-a d←z+a b←a+c a←3 escribe(a,b,z) fin_procedimiento

Algoritmo ejemplo a←1 b←2←1 c←3←5 d←4←8	Funcion fun X←1 y←2←1 c←3 z←3←5	Procedimiento proc (E a←2←3 b←5←7 z←3

Parámetros por valor: x,a

Por referencia:y,z,b

Parámetros actuales: a,b,c//b*2,c

Parámetros formales:x,y,z//a,b

Variables locales: c//z

Variables globales: a,b,c,d

Imprime por pantalla:
1,1,5,3
1,1,5,4
3,7,3
1,1,5,8

Traducción al lenguaje C:

```
#include <stdio.h>
Int a,b,c,d;
Void proc(int a, int *b){
Int z;
z=c-a;
d=z+(*b);
b=a+c;
a=3;
printf("%d %d %d",a,b,z);
}
```

```
Int fun(int x, int *y, int *z) {
Int c;
*y=1;
c=(*y)+a+1;
*z=a+x+c;
Printf("%d%d%d%d",x,*y,*z,
d);
return (c+b);
}
```

```
Int main(){
a=1;
b=2;
c=3;
d=fun(a,&b,&c);
printf("%d%d%d%d",a,b,c,d)
proc(b*2,&c);
printf("%d%d%d%d",a,b,c,d)
return 0;
}
```

Indica la salida del siguiente programa, así como sus parámetros actuales y formales, variables globales y locales, y pasos por valor y por referencia.

```
#include <stdio.h>
#include <stdlib.h>
int x=2;
char y=a;

void a() {
printf("%d\n,x);
x=x*3;
printf("%d\n,x);
printf("%c\n,y);
}

Void b() {
Static int x=2;
x*=5;
Printf("%d\n",x);
y++;
printf("%c\n",y);
}

Void d(int *x, char y){
*x=*x+5;
y+=2;
printf("%d\n",*x);
printf("%d\n",y);
b();
printf("%c\n",y);
a()
}
```

```
int main() {
int x=4;
char y=j;
{
int x=5;
printf("%d\n"x);
}
printf("%d\n"x);
a();
printf("%d\n"x);
b();
printf("%d\n"x);
printf("%c\n"y);
b();
printf("%d\n"x);
printf("%c\n"y);
d(&x,y);
printf("%d\n"x);
printf("%c\n"y);
system("pause");
return 0;
}
```

Globales:	main:	{…}:	b:	d:
x=2=6=18	x=4=9	x=5	x=2=10=50=250	x=4=9
y=a=b=c=d	y=j			y=j=l

Imprime por pantalla: 5, 4, 2, 6, a, 4, 10, b, 4, j, 50, c, 4, 9, l, 250, d, l, 6, 18, d, 9, j

Variables globales: x,y

Variables locales:
-main: x,y
-{…}: x
-b: x

Parámetros actuales:&x,y

Parámetros formales: *x,y

Paso por valor: y

Paso por referencia: x

Escribe una función en C que reciba dos cadenas de caracteres. Debe devolver otra cadena formada por los caracteres de la segunda que se repitan en la primera sin contener caracteres repetidos. Sin utilizar string.h

```
#DEFINE max 512
Void fun(char *vec, char char 1[max], char char2[max]) {
   Int i=0, j=0, k=0, z=0;
   While(char1[i]!=NULL) {
    If(char1[i]==char2[j]) {
       K=0;
       While(z!=1)  {
          If(*vec[k]==NULL) {
             Z++;
             *vec[k]=char2[j];
        }
        If(*vec[k]==char2[j]) {
             z++;
             k--;
        }
     }
    J++;
   }
   I++;
  }
}
```

Implementa una función void ocurrencias(char* cad1, char* cad2)que escriba por pantalla las veces que aparece cada carácter de cad2 en cad1.

```
Void ocurrencias(char *cad1, char cad 2) {
   Int i=o, j, cont=0;
   While(cad2[i]!='\0'){
      J=0;
      Cont=0;
      While(cad1[j]!='\0'){
         If(cad1[j]==cad2[i]) cont++;
      J++;
     }
     Printf("El carácter %c aparece %d veces\n", cad2[i], cont);
     I++;
     }
}
```

Escribe un programa en C que lea la primera palabra de un fichero llamado “mispalabras.txt” y almacene datos en una cadena de caracteres. Se valorará el uso de memoria dinámica.

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

Int tam_1_palabra(FILE *fich); {
   Int cont=0;
   Char c;
   C=fgetc(fich);
   Rewind(fich);
   While((c!=’ ‘)&&(c!=EOF)){
      Cont++;
      C=fgetc(fich);
   }
   Return cont;
}

Int main() {
   FILE *fich;
   Int cont;
   Char *cad, *s;
   Fich=fopen(“mispalabras.txt”,”r”);
   If (fich==NULL) {
      Printf(“Error de apertura”);
      Exit(1);
   }
   Cont=tam_1_palabra(fich);
   Cad=(char *)malloc(sizeof(char)*(cont+1));
   If(cad==NULL) {
      Printf(“Memoria insuficiente”);
      Return 1;
   }
   Rewind fich;
   S=fgets(cad, cont, fich);
   Cad[cont]=’\0’;
   Printf(“%s”,cad);
   Free(cad);
   Fclose(fich);
   System(“pause”);
   Return 0;
}
```

Enunciados sobre estructuras y tipos de datos.

En el curso de un río sin afluentes se suceden 15 pantanos. Cada pantano se encuentra a un nivel determinado según el contenido de agua que posee en cada momento. Todos los pantanos tienen además un nivel de seguridad que puede variar en cada uno, dependiendo de su tamaño, y que en caso de ser superado, provocará el vertido de las aguas sobrantes al río, continuando su curso hasta llegar al siguiente pantano, o al mar si fuese el último. Se quiere realizar una aplicación que controle los vertidos de agua de los pantanos en caso de lluvias localizadas. Se almacenará el nombre de cada pantano, así como los hectómetros cúbicos que tiene de capacidad máxima de seguridad, capacidad mínima, y contenido real. Define en pseudocódigo la estructura de datos necesaria para almacenar dicha información. Suponiendo que la información se encontrase ya almacenada en dicha estructura de datos, escribe una función en pseudocódigo que reciba el nombre de un pantano, cantidad de agua de las lluvias caídas en dicho pantano y muestre las consecuencias de dichas lluvias.

```
Tipo Registro: pantano
   Cadena: nombre
   Entero:actual
   Entero: capacidad
   Entero: seguridad
Fin registro
Vector[15] de pantano: v_pantano

//Cabecera: precipitaciones(E/S pantano: pant)
//precondición: pantanos inicializados.
//postcondicion: realiza los trasvases provocados por la lluvia
Procedimiento precipitaciones(E/S pantano pant)
Var
Cadena: nombre
Entero: a, lluvias, i
Inicio
Escribir("Introduce el nombre de un pantano")
Leer(nombre)
Escribir("Hectómetros cúbicos que llovieron sobre dicho pantano")
Leer(lluvias)
Desde a←1 hasta 15 hacer
Si nombre=pantano[a].nombre entonces
i←a
fin si
fin desde
Desde a←i hasta N hacer
Si (pantanos[a].actual+lluvias>pantanos[a].seguridad) entonces
Pantanos[a].actual←pantanos[a].seguridad
Lluvias←pantanos[a].actual-pantanos[a].seguridad
Si no
Pantanos[a].actual←pantanos[a].actual+lluvias
Lluvias←0
Fin si
Fin desde
Fin procedimiento
```

Un banco tiene 20 trabajadores en cada una de sus 5 sucursales. De cada sucursal se debe almacenar el código y la dirección. Del trabajador se almacena nombre, apellido, DNI, retribuciones complementarias, en cada uno de los 12 meses de trabajo. Las retribuciones son horas extras, plus de producción, y complementos. Diseña la estructura de datos necesaria para almacenar dicha información, y, suponiendo que esté ya inicializada, una función para acceder a las retribuciones de un empleado cualquiera en un mes cualquiera.

```
Const N=15
T=20
M=12
Tipo registro: reg_retribuciones
Entero: horas
Entero: plus
Entero: complemento
Fin registro
Vector[M] de reg_retribuciones: v_retribuciones
Registro: reg_trabajador
Cadena: nombre
Cadena: apellido
Cadena: dni
V_retribuciones: retribuciones
Fin registro
Vector[5] de reg_trabajador: v_trabajador
Registro: reg_sucursal
Entero: código
Cadena: dirección
V_trabajador: trabajador
Fin registro
Vector[N] de reg_sucursal: v_sucursal

Inicio
V_sucursal: empleados

//Cabecera: mostrarempleado(E empleado: trabajador)
//precondición: trabajadores ya inicializados
//postcondicion: muestra las retribuciones de un empleado
Procedimiento mostrar empleado(E empleado: trabajador)
Var
Entero: sucursal, mes empleado
Leer(sucursal)
Leer(empleado)
Leer(mes)
Escribir("El empleado"empleados[sucursal].trabajador[empleado].nombre,
empleados[sucursal].trabajador[empleado].apellido,"con DNI"
empleados[sucursal].trabajador[empleado].DNI,"le corresponden"
empleados[sucursal].trabajador[empleado].retribuciones[mes].plus"por pluses",
empleados[sucursal].trabajador[empleado].retribuciones[mes].horas,"por horas extra, y
"empleados[sucursal].trabajador[empleado].retribuciones[mes].complemento" de
complemento en el mes" mes)
Fin procedimiento
```