UNIVERSIDAD CENTRAL DE VENEZUELA
FACULTAD DE CIENCIAS
ESCUELA DE COMPUTACIÓN
ALGORITMOS Y ESTRUCTURAS DE DATOS

PRÁCTICA DEL TIPO DE DATO APUNTADOR

1. ¿Qué es el tipo de dato referencia o apuntador?
2. Describa el uso que se le da en este contexto a los operadores "&, *"
3. ¿Es posible tener apuntadores de apuntadores y anidarlos cuantas veces sea necesario?
4. Para las siguientes instrucciones, haga un mapa de memoria que muestre que ocurre en ella, y cuál es la salida del programa.

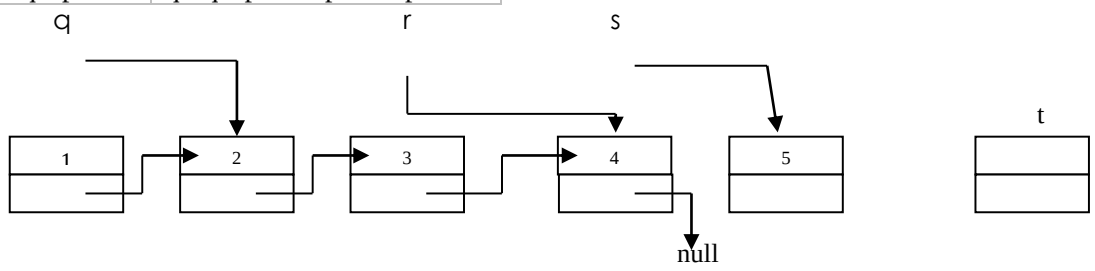
```
struct node {
  int info;
  node *next;
}
int main ( ) {
    node *p, *r, *s;
    p = new node();
    s = new node();
    r = new node();
    *(p).next = r;
    *r.next = s;
    s->next = p;
    s->info = 3;
    p->next->next->next->info = 2;
    *(s->next).next->info = 1;
    p = s->next;
    std::cout << *(p).info << " " << s->info << " " << *(r).info) << std::endl;
    return 0;
}
```

5. Dada las siguientes definiciones:

```
class Nodo {
  public:
   int N;
   Nodo *prox;
}
Nodo *q, *r, *s;
Nodo t;
```

Asuma como estado inicial la figura que a continuación se muestra, e indique el estado final después de ejecutar cada una de las siguientes instrucciones independientemente y luego una tras otra secuencialmente.

q = q->prox	s->prox= s
*q = *(q->prox)	t = *q
q->prox	*q = *s
q ->prox->prox	*s = t
q = r->prox	q = t
*q = *(*r.prox)	(*r).prox =q
s->prox = q->prox	q= q->prox->prox->prox



6. Dada la siguiente secuencia de instrucciones indique que es lo que ocurre en cada línea. Indique si queda algún espacio de memoria por liberar.

```
class Nodo {
  public:
    int Info;
    Nodo *prox;
}
Nodo *P, *Q;
int *E;
int I;
int **F;

int main ( ) {
  P = new Nodo ( );
  E = new int ( );
  *E = 0;
  F = &E;
  P-> prox = new Nodo ( );
  Q = &(*P);
  P = P->prox;
  Q->Info = 30;
  (*P). Info = Q->Info + 10;
  P->prox = new Nodo( );
  P->prox->Info = Q->Info + P->Info + 10;
  P = (*P).prox;
  P->prox = NULL;
  while ( Q !=  NULL ) {
    *E = *F + Q->Info;
    Q = Q->prox;
  }
  F = new int* ( );
  *F = new int ( );
  **F = 1;
  delete *F;
  *F = &I;
   I=5;
  std::cout << **F  <<  std::endl;
  return 0;
}
```

7. Considere las siguientes declaraciones:

```
int *X, *Y, *Z;
char *W;
int A;
char B;
bool C;
```

Indique el efecto de las siguientes operaciones:

- X = **new int ( )**;
- Y = **new int ( )**;
- W = **new char ( )**;
- X= Y;
- B = *W;

- Z = **new int ( )**;
- Z = W;
- C= W == Z;
- X = **new int ( )**
- *X = 1;
- *W = 'G';
- A = *X + *Y;
- C = (*W == A);
- *Z = A;
- Z= X;
- **delete** Y;

8. Dada la siguiente secuencia de instrucciones indique que ocurre en cada línea. Indique además si queda algún elemento por liberar de memoria al terminar LOL( ).

```
class Nodo {
   public:
   int Info;
   Nodo *prox;
}
Nodo *P, *Q;

void LOL( ) {
  P = new Nodo ( );
  P->Info = 10;
  P->prox = P;
  Q = new Nodo ( );
  P->prox = Q;
  Q->Info = P->Info + 3;
  Q->prox = P;
  Q->prox = NULL;
  Q = new Nodo ( );
  Q->Info = P->Info + P->prox->Info;
  P->prox->prox = Q;
  Q->prox = NULL;
  Q = P;
  while ( Q != NULL ) {
    std::cout  << Q->Info << std::endl;
    Q = Q->prox;
  }
}
```

9. Haga la traza del siguiente algoritmo y explique que sucede en cada instrucción.

```
int main ( ) {
  int i, j, n;
  int *arr;
  int **mat;
  std::cin >> n;
  arr = new int [n];
  mat = new int *[n];
  for ( i=0; i < n -1; i++ ) {
    arr[i] =  i;
    mat[i] = new int[n];
    for ( j=0; j < n -1; j++) {
      mat[i][j] = i + j;
```

```
        }
      }
    i = n – 1;
    while ( i>=0 ) {
      std::cout << arr[i] << std::endl;
      delete []mat[i];
      i = i - 1;
    }

    delete []arr;
    delete []mat;
    // en este momento, ¿a quién apunta arr?. ¿Podría acceder arr[1]?
  return 0;
}
```

10. Realice la traza del siguiente algoritmo. En cada línea, muestre el estado de las estructuras de datos gráficamente, y en caso de que la instrucción sea incorrecta, indicar el tipo de error.

```
    struct Nodo {
      int Info;
      Nodo *prox;
    }

    Nodo **p, **s, *q, *r;
    p = &q;
    q->info = 30;
    q->prox = NULL;
    r =  new Nodo ( );
    r->info = 2;
    s = &r;
    delete q;
    q =  new Nodo ();
    q->info = 31;
    s->prox = q;
    (*p)->info = q->info + (*s)->prox->info;
```

11. Haga la traza del siguiente algoritmo y explique que sucede en cada instrucción.

```
class Point {
    public:
      float x,y;
      Point () {
        x = 0;
        y = 0;
      }
      Point (float x, float y) {
        this-> x = x;
        this->y = y;
      }
};
class Rect {
    public:
    Point *p1, *p2;
    Rect () { p1 = p2 = NULL; }
```

```
        Rect (Point *p1, Point *p2) {
          this->p1 = p1 ;
          this->p2 = p2 ;
        }
        ~Rect() {
          if (p1 !=  NULL) delete p1 ;

          if (p2 != NULL) delete p2;
        }
  };
  int main() {
    Point *a, *b;
    a = new Point ();
    b = new Point (1,1);
    Rect *r = new Rect (a,b);
    delete a;
    delete r;      // ¿Qué error ocurre aquí??
    Rect otro(a,b);
    return 0;
  } // ¿Qué sucede al llamar al destructor de otro??
```

12. Indique el estado final, y responda las preguntas comentadas en Any ().

```
struct node {
  int info;
  node *prox;
}
void X (Nodo *&q, int value) {
   q = new Nodo ();
   q->info = value;
   q->prox = NULL;
}
void Y (Nodo **q, int value) {
   *q = new Nodo ();
   *q->info = value;
   *q->prox = NULL;
}
void W (Nodo *q, int value) {
   q->info = value;
}
void Z (Nodo *q, int value) {
   q = new Nodo ();
   q->info = value;
   q->prox = NULL;
}
void  Any() // Principal
  Nodo  *p;
  X (p,1);
  Y (p->prox, 2);
  Z (p->prox->prox, 3);
  W (p,4);
}
// a) ¿Cuál es el estado de la memoria en este momento?
// b) ¿Es posible liberar todos los elementos creados con New? ¿De qué forma?
```