# CC3301
# Programación de Software de Sistemas
# Profesor: Luis Mateu

- Declaración de punteros con inicialización
- Cómo cambiar parámetros
- Punteros a punteros
- Punteros a punteros a estructuras
- Punteros a funciones: invocación y declaración
- Typedef para punteros a funciones

# *Declaración de punteros con inicialización*

- Considerando esta declaración:

  int a, *p = &a;

- ¿Qué variable se está inicializando? ¿*p* o *a*?
- Respuesta: *¡ p !*
- La sintaxis es engañosa: ¿por qué?
- El significado de * cambia según el contexto:
  a) como operador binario es la multiplicación: a * b
  b) como operador unario es el operador de contenido: *p
  c) En una declaración, un * antes del identificador que se está declarando es un modificador de tipo, no es el operador de contenido, señala que el identificador es un puntero
- La declaración int *p significa que cuando aparezca *p en una expresión, su tipo será int
- ¿Cuando vale c? int a=10, *p= &a, c=*p*a;

Expresión

# *Cómo cambiar parámetros*

- Función que intercambia valores de 2 variables

```
// Versión incorrecta
void swap(int x, int y) {
  int tmp= x;
  x= y;
  y= tmp;
}  // ¡No haga esto!
```

```
// Uso
int main() {
    int a= 1, b= 2;
    swap(a, b);
    printf("%d %d\n", a, b);
} // Resultado: 1 2
```

```
// Versión correcta
void swap(int *px, int *py) {
    int tmp= *px;
    *px= *py;
    *py= tmp;
}
```

```
// Uso
int main() {
    int a= 1, b= 2;
    swap(&a, &b);
    printf("%d %d\n", a, b);
} // Resultado: 2 1
```

- ¿Función que intercambia valores de 2 punteros?

```
    int a= 1, b= 2, *p= &a, *q= &b;
    swap_ptr(&p, &q);
    printf("%d %d %d %d\n", a, b, *p, *q);
    // Esperado:1 2 2 1
```

# *Punteros a punteros*

- Función que intercambia valores de 2 variables

```
// Versión correcta
void swap_ptr(int **px, int **py) {
    int *tmp= *px;
    *px= *py;
    *py= tmp;
}
```

- *Un puntero a un puntero es una variable que contiene direcciones de punteros*
- El modificador de tipo en la declaración es **
- ¿Cual es el tipo de la expresión *px?

- Respuesta: int * porque la declaración int * *px dice que en una expresión el tipo de *px es int *

# *Punteros a punteros a estructuras*

- Función que elimina el primer nodo de una lista simplemente enlazada

```
void elim(Nodo *cabeza) {
    Nodo *rem= cabeza;
    cabeza= cabeza->prox;
    free(rem);
} // Versión incorrecta
```

```
// Uso
Nodo *h= ...;
elim(h); ✕
// h es dangling reference
```

```
void elim(Nodo **pcabeza) {
    Nodo *cabeza= *pcabeza;
    *pcabeza= cabeza->prox;
    free(cabeza);
} // Versión correcta
```

```
// Uso
Nodo *h= ...;
elim(&h);
```

- Ejercicio: función que agrega un nodo al comienzo de una lista simplemente enlazada

```
// Uso
Nodo *h= ...;
agregar(&h, "gato");
```

# *Punteros a funciones: Motivación*

- Función que calcula numéricamente la integral de *f* usando el método de los trapecios.
- Se puede aproximar el área de la curva como la suma de las áreas de *n* trapecios. Esta es la fórmula:

$$\int_{xi}^{xf} f(x)\,dx \approx h\cdot\left[\frac{f(xi)+f(xf)}{2}+\sum_{k=1}^{n-1} f(xi+k\cdot h)\right] \quad \text{con} \quad h=\frac{xf-xi}{n}$$

- Implementación:

```
double integral(double xi, double xf, int n) {
   double h= (xf-xi)/n;
   double sum= ( f(xi) + f(xf) ) / 2;
   for (int k= 1; k<n; k++)
      sum += f(xi + k*h);
   return sum * h;
}
```

- Si ahora se necesita la integral de g hay que reescribir la función integral cambiando *f* por *g*.
- Idea: pasar la función como parámetro.

# *Punteros a funciones: invocación*

- El encabezado para la función *f* es:

```
double f(double x);
```

- El identificador *f* representa la dirección de la primera instrucción de máquina de la función
- ¿Cómo se declara un puntero *pf* que almacena la dirección de una función?
- Primero hay que pensar en cómo se invoca la función almacenada en *pf*
- Los diseñadores de C argumentaron que si se escribe **p* para usar la variable a la cual apunta un puntero *p*, entonces debería ser lo mismo para invocar la función almacenada en *pf*:

```
double x= 5.0,   fx= *pf(x);  // No sirve: * ( pf(x) )
```

- Problema: el operador ( ) tiene mayor precedencia que *
- Solución: parentizar

```
double x= 5.0,   fx= (*pf)(x);  // ¡Forma correcta!
```

# *Declaración de un puntero a una función*

- El encabezado para la función *f* y *g* es:

```
double f(double x),    g(double x);
```

- Dice que en una expresión el tipo de la invocación f(x) es double, en donde x es de tipo double
- La declaración del puntero *pf* es:

```
double (*pf)(double x);
```

- Dice que en una expresión el tipo de la invocación (*pf)(x) es double, en donde x es de tipo double
- Entonces:

```
pf= f;                          // pf apunta a f
double fx5= (*pf)(5.0);         // Invoca f(5.0)
pf= g;                          // pf apunta a g
double gx2= (*pf)(2.0);         // Invoca g(2.0)
```

- *pf* no debe apuntar a funciones con un encabezado distinto del de *f* y *g*

# *Función integral genérica*

- Implementación:

```
double integral(double (*pf)(double x),
                        double xi, double xf, int n) {
   double h= (xf-xi)/n;
   double sum= ( (*pf)(xi) + (*pf)(xf) ) / 2;
   for (int k= 1; k<n; k++)
      sum += (*pf)(xi + k*h);
   return sum * h;
}
```

- ¡Horrible!

```
double poli(x) {      // Ejemplo de uso
   return 4.5*x*x-10*x+3.1;
}
int main( ) {
   printf("%f\n", integral(poli, 0.0, 10.0, 100));
   return 0;
}
```

# *Typedef para punteros a funciones*

- ¿Qué pasa si agregamos typedef a la declaración de un puntero a una función?

```
typedef double (*Fun)(double x);
```

- Fun ya no es puntero a una función: es el tipo de los punteros a funciones que reciben un parámetro real y retornan un real
- La función integral se puede reescribir:

```
double integral(Fun pf,
                double xi, double xf, int n) {
   …
}
```

# *Más integrales*

- Se desea programar la función:

  double h(double a, double b, double c,
  double xi, double xf, int n);

- tal que:

$$h(a,b,c,xi,xf,n) \approx \int_{xi}^{xf} ax^2 + bx + c\,dx$$

  Aproximado con *n* trapecios

- ¿Se puede usar integral? ¿Hay manera de pasarle los valores de *a*, *b* y *c*?

- Se desea programar la función:

  double (*Fun2) (double x, double y);

  double e(double xf, double yf, Fun2 g, int n);

- tal que:

$$e(xf,yf,g,n) \approx \int_{0}^{yf}\int_{0}^{xf} g(x,y)\,dx\,dy$$

  Aproximado con *n* trapecios

- ¿Sirve integral?

# *Solución incorrecta para h*

```
double poli2(double x) {
   return a*x*x+b*x+c;  // ✕
}
double h(double a, double b, double c,
                double xi, double xf, int n) {
   return integral(poli2, xi, xf, n);
}
```

- Problema: No compila
- Las variables *a*, *b* y *c* no son *visibles* en *poli2*