Clase #15 de 29
Punteros & Arreglos

Ago 5, Martes
Ago 13, Miércoles

K&R – Capítulo 5

Punteros y Arreglos

Punteros y Direcciones

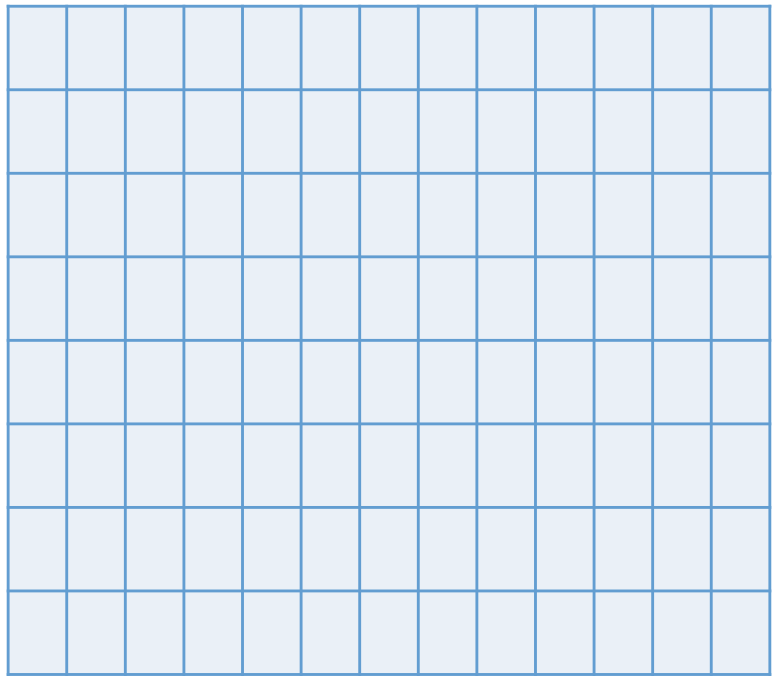
- Puntero
  - Variable que contiene dirección de una variable
  - Muy relacionados con arreglos
- ¿Por qué usarlos?
  - A veces, única forma
  - Código compacto y eficiente
  - Claridad y simplicidad
- Operadores unarios
  - & Dirección de
  - * Indirección o Desreferencia
- Dos formas de leer la declaración de punteros
  - ¿Qué tipo tiene *p?
  - ¿Qué tipo tiene p?

```
{
   char *p;
   char c='A';

   p = &c;
}
```

	p						c		
	1038						65		
1031	1032	1033	1034	1035	1036	1037	1038	1039	1040

Punteros y Direcciones (cont.)

```
{
  int x=1, y=2, z[3];
  int *ip, *iq; //pointer to int
                //int* ip, iq
  ip = &x;      //ip now points to x
  y = *ip;      //y is now 1
  *ip = 0;      //x is now 0
  ip = &z[0];   //ip now points to z[0]


  *ip = *ip + 2;// x y *ip
  *ip += 2 ;
  ++*ip ;
  (*ip)++ ;

  iq = ip ;
}

double *dp, atof(char *);
```



Punteros como argumentos

```
void swop(int, int);
void swap(int *, int *);

int main(void){
    int a=17, b=39;
    swop(  a, b  );
    printf("%d %d\n", a, b);
    swap( &a, &b );
    printf("%d %d\n", a, b);
}

void swop(int x, int y) {
    int temp;
    temp = x;
    x = y;
    y = temp;
}

void swap(int *px, int *py){
    int temp;
    temp = *px;
    *px = *py;
    *py = temp;
}
```

	0	1	2	3	4	5	6	7	8	9
0		17				39				
1										
2										
3										
4										
5										
6										

Punteros y Arreglos

```
int a[3];
 a[i]
int *p;
p = &a[0];
int x;
x = *p;
p + i
*( p + i )
a[i]
a
p = &a[0]
p = a
 a[i] ==  *(a+i)
&a[i] ==  a+i
 p[i] ==  *(p+i)
p ¿=? a
p++ p=a
a++ a=p
```

	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										

- Están muy relacionados
- La subindicación se puede lograr con punteros, a veces, más eficientemente
- Aritmética de punteros
- Semántica de a
- Expresión subindicación es equivalente a expresión puntero y offset
- Diferencia
  - Objeto al que refieren

Aritmética de Direcciones

```
Tipo *p;

p++

p+=i
```

- Independiente del tipo de dato
- Operaciones sin semántica
  - Sumar dos punteros
  - Multiplicar
  - Dividir
  - Correr
  - Enmascarar
  - Sumar flotantes a punteros
  - Asignación entre punetos a diferentes tipos (excepto void)

Operaciones con Semántica y Otra versión de Strlen

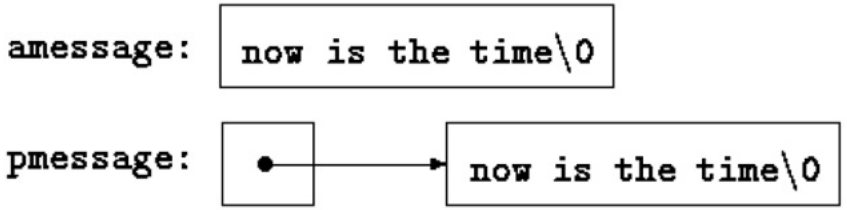
- Operaciones con semántica
  - Asignaciones entre punteros al mismo tipo sin cast
  - Comparaciones por igualdad
  - Otras comparaciones dentro del arreglo
  - Suma y resta con enteros
  - Resta entre punteros
    - p-q+1
  - Asignación y comparación con cero

```
size_t strlen(const char *s){
 const char *p = s;

 while(*p != '\0')
  p++;

 return p - s;
}
```

Síntesis de Arreglos, Punteros, y Funciones

Puntero a Caracteres y Funciones

```
char amessage[] = "now is the time"; /*an array*/
char *pmessage = "now is the time"; /*a pointer*/
```




```
char a[4+1], b[]="hola", *c="chau";

a=b    a=c    b=c    c=b   // ¿Cúales son válidas?
```

```
// strcpy: copy t to s; array subscript version
void strcpy(char *s, char *t){
    int i;
    i = 0;
    while ((s[i] = t[i]) != '\0')
        i++;
}
```

```
// strcpy: copy t to s; pointer version 2
void strcpy(char *s, char *t){
    while ((*s++ = *t++) != '\0')
    ;
}
```

```
/. strcpy: copy t to s; pointer version
void strcpy(char *s, char *t)
{
    int i;
    i = 0;
    while ((*s = *t) != '\0') {
        s++;
        t++;
    }
}
```

```
/* strcpy: copy t to s; pointer version 3,
the Idiom */
void strcpy(char *s, char *t){
    while ( *s++ = *t++ )
        ;
}
```

```
<string.h>
char *strcpy(char *s, const char *t);
```

Arreglos en Expresiones: La Regla y las Excepciones

- La regla: Una **expresión** de **tipo** "arreglo de *tipo*" se **convierte** en una **expresión**
- con **tipo** "puntero a *tipo*",
- con **valor** dirección del primer elemento del arreglo y
- no es un **valor-l**

- Aplicaciones de la regla en azul

```
1.  int a[]={1,2,3}, *pa, i=2;
2.  char s[]="abcd", *ps;
3.  a // &a[0]
4.  s // &s[0]
5.  a[i] ≈ *(a+i) ≈ *(i+a) ≈ i[a] // 3
6.  s[i] ≈ *(s+i) ≈ *(i+a) ≈ i[a] // 'c'
7.  pa=a, ps=s
8.  pa[i] ≈ *(pa+i) ≈ *(i+pa) ≈ i[pa]// 3
9.  ps[i] ≈ *(ps+i) ≈ *(i+ps) ≈ i[ps]//'c'
10. void f(int*);       // recomendada
11. // void f(int[]);   // equivalente
12. // void f(int[N]); // equivalente
13. void g(char*);      // recomendada
14. // void g(char[]); // equivalente
15. // void g(char[N]);// equivalente
16. f(a), f(pa) , g(s) , g(ps)
17. ps="YXZ"
18. ps[i] ≈ *(ps+i) ≈ *(i+ps) ≈ i[ps]              // 'Z'
19. "YXZ"[i] ≈ *("YXZ"+i) ≈ *(i+"YXZ") ≈ i["YXZ"] // 'Z'
```

- Las excepciones a la regla: En las siguientes situaciones no se aplica la regla de conversión de tipo de la expresión
- (1) cuando es la **cadena literal** que **inicializa** un **arreglo**
- ó (2) Cuando es el operando del **operador sizeof**
- ó (3) del **operador unario &**

- Aplicaciones de la regla en azul, excepciones en rojo

```
18. int a[]={1,2,3,4};
19. char *ps="wxyz", s[]="abcd";

20. sizeof ps     //sizeof(char*)
21. sizeof a      //sizeof(int[4]) ≈ sizeof(int)*4
22. sizeof s      //sizeof(char[5])≈ sizeof(char)*5 ≈ 1*5 ≈5
23. sizeof "1234" //sizeof(char[5])≈ sizeof(char)*5 ≈ 1*5 ≈5

24. //Expr     Type       Valores ejemplo de direcciones
25. a       // int*       100
26. s       // char*      132
27. "1234"  // char*      137
28. &a      // int(*)[4]  100
29. &s      // char(*)[5] 132
30. &"1234" // char(*)[5] 137.
```

¿Consultas?

Fin de la clase