

Tipos: register, extern
Preprocesador: Macros
Punteros y Arreglos

Variables tipo registro

- Útil para variables que se utilizan intensamente:
- Estas variables se colocan en registros de la máquina:
 - Programas más pequeños y rápidos
- Declaración:


```
register int x;
register char c;
```

Variables tipo registro

- También se pueden utilizar en funciones:


```
func (register unsigned m, register long n){
    register int i;
    ...
}
```
- Restricciones según hardware disponible

Variables tipo registro

- Compiladores tienen libertad de ignorarlas:
 - Exceso no hace daño
- ¿Se puede acceder a la dirección de una variable tipo "register"?

Variables Externas

- Variables declaradas fuera de las funciones:

```
int x;
f(){
    extern x;
    /* aquí se puede usar x */
}
```

- Potencialmente disponibles para cualquier función

Variables Externas

- Son una alternativas a los argumentos de las funciones:
 - Preferibles a largas listas de argumentos
- Son permanentes, tienen mayor alcance y tiempo de vida
- Ejemplo de funciones que comparten datos: push() y pop()

Preprocesador de C

- Elementos más usados:
 - *#include: incluye el contenido de un archivo durante la compilación*
 - *#define: reemplaza un símbolo por una secuencia arbitraria de caracteres*
- Cuando se cambia un archivo “incluido” se debe recompilar

Preprocesador de C

- Para #define con un texto de varias lineas, se utiliza “\” al final de cada línea.
- Las sustituciones se realizan sólo para elementos (ej: no en printf)
- Cualquier nombre puede definirse con cualquier texto de reemplazo:
 - #define porsiempre for(;;)

Preprocesador de C

- Macros con argumentos:

```
#define max(A, B) ((A) > (B) ? (A) : (B))
```
- El uso de “max” se expande a código:

```
x = max(p+q, r+s)
```

 se reemplaza por:

```
x = ((p+q)>(r+s)?(p+q):(r+s));
```
- “max” sirve para cualquier tipo de datos, a diferencia de una función

Preprocesador de C

- Riesgos de utilizar “max”:

```
max(i++, j++);
```
- El valor más grande se incrementará 2 veces
- Cuidado con los paréntesis:

```
#define cuadrado(x) x*x
```

```
cuadrado(z+1)
```

Preprocesador de C

- Inclusion condicional:

```
#if .....  
#elif .....  
#else .....  
#endif
```
- Incluir un archivo sólo una vez:

```
#if !defined(HDR)  
#define HDR (contenido del archivo)  
#endif
```

Punteros y Arreglos

- Operador & entrega la dirección de memoria de un objeto
- Operador * da acceso al objeto que señala el puntero

```
int x = 1, y = 2, z[10];  
int *ip;    // puntero a int  
ip = &x;    // ip apunta a x  
y = *ip;    // y toma el valor 1  
*ip = 0;    // x toma el valor 0  
ip = &z[0] // ip apunta a z[0]
```

Punteros y Arreglos

- `*ip` es un `int`:

```
*ip = *ip + 10; // incrementa *ip en 10
y = *ip + 1; // (lo apuntado por ip) + 1
*ip += 1; // incr. en 1 lo apuntado por ip
++*ip; // idem
(*ip)++; // idem
```
- Operadores `*` y `++` se asocian de derecha a izquierda

Punteros y Arreglos

```
int a[10];
int *p;
p = &a[0]; /* p apunta al primer elemento de a */
printf("%d\n", *(p+1)); /* imprime a[1] */
printf("%d\n", *(p+i)); /* imprime a[i] */
```

- Además:

```
p = &a[0]; // es equivalente a:
p = a;     // por lo tanto:
*(a+i) es equivalente a a[i]
```

Punteros y Arreglos

- Ejemplo: cálculo del largo de un string

```
int my_strlen(char *s){
    int n;
    for(n=0; *s!='\0'; ++s)
        ++n;
    return n;
}
main(){
    printf("%d\n", my_strlen("hola"));
}
```

Punteros y Arreglos

- Implementación con aritmética de punteros:

```
int my_strlen(char *s){
    char *p=s;
    while(*p!='\0')
        ++p;
    return p-s;
}
main(){
    printf("%d\n", my_strlen("hola"));
}
```

Punteros y Arg. Funciones

- Ejemplo de rutina swap:

```
void swap(int x, int y){  
    int tmp = x;  
    x = y;  
    y = tmp;  
}
```

¿Intercambia los valores?
(swap.c)

Punteros y Arg. Funciones

- Se debe hacer con punteros:

```
void swap(int *x, int *y){  
    int tmp = *x;  
    *x = *y;  
    *y = tmp;  
}
```

(swap2.c)