

Sistemas Operativos

Problemas frecuentes con el lenguaje C

Grado en Ingeniería Informática

Universidad Carlos III de Madrid



Contenidos

- Errores comunes:
 - Generales
 - Tiras de caracteres
 - Entrada y salida
- Problemas con punteros:
 - Punteros
 - Punteros en paso de parámetros
 - Punteros como resultado
 - Punteros y arrays
 - Punteros a funciones



Contenidos

- Errores comunes:
 - **Generales**
 - Tiras de caracteres
 - Entrada y salida
- Problemas con punteros:
 - Punteros
 - Punteros en paso de parámetros
 - Punteros como resultado
 - Punteros y arrays
 - Punteros a funciones

Errores comunes: generales

- Olvidar poner un break en un switch

```
int x = 2;

switch(x) {
    case 2: printf("Two\n");
    case 3: printf("Three\n");
}
```

```
int x = 2;

switch(x) {
    case 2: printf("Two\n");
            break;
    case 3: printf("Three\n");
            break;
}
```

Errores comunes: generales

- Usar = en lugar de ==

```
int x = 5;

if ( x = 6 )
    printf("x equals 6\n");
```

```
int x = 5;

if ( x == 6 )
    printf("x equals 6\n");
```

Errores comunes: generales

- Tamaño de los vectores

```
int a[10];
```

```
a[10] = ...
```

```
int a[10];
```

```
a[0] = ...
```

```
...
```

```
A[9] = ...
```

Errores comunes: generales

- Poner un ; extra al final de la cabecera de un bucle

```
int x = 5;  
while( x > 0 );  
    x--;
```

```
int x = 5;  
while( x > 0 )  
    x--;
```



Contenidos

- Errores comunes:
 - Generales
 - **Tiras de caracteres**
 - Entrada y salida
- Problemas con punteros:
 - Punteros
 - Punteros en paso de parámetros
 - Punteros como resultado
 - Punteros y arrays
 - Punteros a funciones

Errores comunes: tira de caracteres

- Comparar usando ==

```
char st1[] = "abc";  
char st2[] = "abc";  
if ( st1 == st2 )  
    printf("Yes");  
else  
    printf("No");
```

```
char st1[] = "abc";  
char st2[] = "abc";  
if ( strcmp(st1,st2) == 0 )  
    printf("Yes");  
else  
    printf("No");
```



Contenidos

- Errores comunes:
 - Generales
 - Tiras de caracteres
 - **Entrada y salida**
- Problemas con punteros:
 - Punteros
 - Punteros en paso de parámetros
 - Punteros como resultado
 - Punteros y arrays
 - Punteros a funciones

Errores comunes: entrada/salida

- Dejar caracteres en el buffer de entrada

```
int x;  
char st[31];  
  
printf("Enter an integer: ");  
scanf("%d", &x);  
printf("Enter a line of text: ");  
fgets(st, 31, stdin);
```

```
int x; int ch;  
char st[31];  
  
printf("Enter an integer: ");  
scanf("%d", &x);  
while( (ch = fgetc(fp)) != EOF && ch != '\n' );  
printf("Enter a line of text: ");  
fgets(st, 31, stdin);
```



Contenidos

- Errores comunes:
 - Generales
 - Tiras de caracteres
 - Entrada y salida
- Problemas con punteros:
 - **Punteros**
 - Punteros en paso de parámetros
 - Punteros como resultado
 - Punteros y arrays
 - Punteros a funciones

Cuidado en la compilación

- ¡Ojo con los *warnings*!
- Suelen avisar de usos y asignaciones erróneas (incluyendo el posible mal uso de punteros)
- Por eso es importante que los programas no tenga *warnings* (advertencias)

Cuidado en la ejecución

- Si el programa funciona unas veces pero falla de vez en cuando, de forma imprevisible y se usan punteros, entonces:
 - probablemente hay error(es) en el uso de punteros
- Interesante buscar una secuencia de prueba en la que siempre falle, como caso útil para depurar.

Punteros (1/4)

- Dos variables

- `int *px;` //Tiene una pinta así:
`0x00aabbff`
- `int x;` //Tiene una pinta así: 5

- ***px = x;**

- **MAL:** Si no se ha reservado memoria.
- **BIEN:** Si sí se ha reservado memoria.
- El siguiente código está mal:
`int *px;`
`*px = 5;`

Punteros (2/4)

- **px = x;**
 - **MAL:** Estamos asignando un entero a una dirección de memoria.
- ***px = &x;**
 - **MAL:** Estamos asignando una dirección de memoria a un entero (el contenido).
- **px = &x;**
 - **BIEN:** px apunta ahora a la dirección de x.
 - Podemos trabajar con px (`*px = *px + 1`) y los cambios se harán sobre x.

Punteros (3/4)

- **x = px;**
 - **MAL:** Estamos asignando una dirección de memoria a un entero.
- **&x = px;**
 - **MAL:** Operador de dirección siempre a la derecha.
 - Esta operación implicaría un movimiento de memoria (de ser válida).
- **x = *px**
 - **BIEN:** Asignamos a x el contenido de la dirección apuntada por px.

Punteros (4/4)

- **x = &px;**
 - **MAL:** Estamos asignando una dirección de memoria (puntero a puntero) a un entero.
 - Aplicar el operador de dirección sobre un puntero nos da un puntero a puntero.
 - El siguiente código está bien:

```
int **y;  
y = &px;
```



Contenidos

- Errores comunes:
 - Generales
 - Tiras de caracteres
 - Entrada y salida
- Problemas con punteros:
 - Punteros
 - **Punteros en paso de parámetros**
 - Punteros como resultado
 - Punteros y arrays
 - Punteros a funciones

Punteros en paso de parámetros (1/2)

- La siguiente forma de utilizar los punteros es muy común: variables locales y operador de dirección.

```
void funcion1(struct *par1, int *par2){  
  ..  
}
```

```
void funcionPadre(){  
  struct var1;  
  int var2;  
  
  funcion1(&var1, &var2);  
}
```

Punteros en paso de parámetros (2/2)

- La siguiente forma de utilizar los punteros es muy común: memoria dinámica y puntero.

```
void funcion1(struct *par1, int *par2){  
    ..  
}
```

```
void funcionPadre(){  
    struct *var1;  
    int *var2;  
  
    var1 = malloc(sizeof(struct));  
    var2 = &otroentero;  
  
    funcion1(var1, var2);  
    free(var1);           //IMPORTANTE LIBERAR LA MEMORIA DESPUES DEL USO  
}
```



Contenidos

- Errores comunes:
 - Generales
 - Tiras de caracteres
 - Entrada y salida
- Problemas con punteros:
 - Punteros
 - Punteros en paso de parámetros
 - **Punteros como resultado**
 - Punteros y arrays
 - Punteros a funciones

Punteros como resultado

- La siguiente forma de utilizar los punteros es muy común: memoria dinámica y puntero.

```
struct x * void funcion1(void){  
    struct x var1;  
    ..  
    return &var1;  
}
```

```
void funcionPadre(){  
    struct *var;  
  
    var =      funcion1();  
    // var es un puntero a una zona de la pila que ya no se usa  
}
```

Punteros como resultado

- La siguiente forma de utilizar los punteros es muy común: memoria dinámica y puntero.

```
struct x * void funcion1(void){
    struct x *var1;
    var1 = malloc(sizeof(struct x)) ;
    ..
    return var1;
}

void funcionPadre(){
    struct *var;

    var =      funcion1(); // la memoria de var a de
                        // ser liberada...
}
```




Contenidos

- Errores comunes:
 - Generales
 - Tiras de caracteres
 - Entrada y salida
- Problemas con punteros:
 - Punteros
 - Punteros en paso de parámetros
 - Punteros como resultado
 - **Punteros y arrays**
 - Punteros a funciones

Punteros y arrays (1/3)

- Arrays: vectores de elementos de tamaño estático.
- Interoperables con punteros.

- Añadimos una variable más a las dos anteriores

– int ax[TAM];	//Tiene una pinta así:	{2, 4, 5, 2}	
• ax	//Dirección del vector:	0x00112244	iguales
• ax[2]	//Elemento 2 del vector:	5	
• &(ax[0])	//Dir del elemento 0 del vector:	0x00112244	
• &(ax[2])	//Dir del elemento 2 del vector:	0x0011224c	

Punteros y arrays (2/3)

- `x = ax[n];`
 - **BIEN:** x toma el valor del dato en la posición n del vector.
- `px = ax[n];`
 - **MAL:** Estamos asignando un entero (elemento n) a una dirección de memoria.
- `*px = ax[n];`
 - **BIEN:** Estamos guardando un entero (elemento n) en la dirección de memoria apuntada por el puntero px.
- `x = ax;`
 - **MAL:** ax representa la dirección de memoria del vector.
 - Por tanto, estamos asignando una dirección de memoria a un entero.

Punteros y arrays (3/3)

- **px = ax;**
 - **BIEN:** px apunta ahora a la dirección de ax.
 - Puesto que la dirección del vector COINCIDE con la dirección del primer elemento del vector entonces:
 - px* y ax[0] son iguales. El contenido de px es el 1er elemento.
- **px = &(ax[n]);**
 - **BIEN:** px apunta ahora a la dirección del elemento n de ax.
 - px = &(ax[3]); //px apunta al elemento 3
//px* representa el valor del elemento 3
 - Si n = 0 entonces:
 - px es igual a &(ax[0]) que es igual también a ax



Contenidos

- Errores comunes:
 - Generales
 - Tiras de caracteres
 - Entrada y salida
- Problemas con punteros:
 - Punteros
 - Punteros en paso de parámetros
 - Punteros como resultado
 - Punteros y arrays
 - **Punteros a funciones**

Punteros a funciones

- Se utilizan para pasar una función como parámetro a otra función
- Es frecuente que algunas estructuras almacenen punteros a funciones, para formar algo parecido a un objeto en java (atributos + métodos).
- El nombre de una función es un puntero a la zona de memoria donde comienza su código
 - `void funcion1(){...}; // función normal`
 - `Funcion1` // Dirección de la función: `0x00112288`

Punteros a funciones

- `void funcion1(){...}`
 - Función normal y corriente
- `void (*punteroAFuncion)(void)`
 - Puntero a función que no retorna nada y no admite parámetros
- `void (*punteroAFuncion2)(void *)`
 - Puntero a función que no retorna nada y con un parámetro puntero genérico
- `float (*punteroAFuncion3)(int, int)`
 - Puntero a función que retorna float y recibe dos parámetros de tipo int

- `punteroAFuncion = funcion1`
 - **BIEN**: El tipo de `funcion1` encaja con el tipo de `punteroAFuncion`.
- `punteroAFuncion2 = funcion1`
 - **MAL**: El tipo de `funcion1` NO encaja con el tipo de `punteroAFuncion2`.
- `punteroAFuncion3 = funcion1`
 - **MAL**: El tipo de `funcion1` NO encaja con el tipo de `punteroAFuncion3`.

Punteros a funciones

paso de una función como parámetro

```
float funcionPasada (int par1, int par2) { return (float)(par1+par2) ;  
    //Función que va a ser pasada como parámetro  
}  
  
void funcionPadre(){  
    float (*punteroAFun)(int, int); //Declaramos el puntero a función  
    int var1, var2;  
    ...  
    punteroAFun = funcionPasada;      //Lo asignamos con la función que  
    deseamos  
    funcionHija(punteroAFun, var1, var2); //Llamamos con el puntero como  
    parámetro  
}  
  
void funcionHija(float (*funcionQueMePasan)(int, int), int par1, int par2){  
    float f;  
    ...  
    f = funcionQueMePasan(par1, par2);      //Invocamos la función pasada  
    como parámetro  
}
```


Punteros a funciones

paso de una función en una estructura

```
float funcionPasada(int par1){    // Función que va a ser pasada como parámetro
    ...
}

typedef struct{                  // Estructura que va a transportar a la función y al
    parametro                    // parámetro
    float (*funcionContenida)(void *);
    void *par1;
} tipoEstructura;

void funcionPadre(){            // Función padre que inicializa la estructura
    struct variableEstructura tipoEstructura;
    int x = 5;
    variableEstructura.funcionContenida = funcionPasada; // Asignación de la
                                                    // función pasada al puntero
    variableEstructura.par1 = (void *)&x;
    funcionHija(&variableEstructura);
}

void funcionHija(tipoEstructura *estructuraPasada){
    float f;
    ...
    f = estructuraPasada->funcionContenida((int)(estructuraPasada->par1));
}
```

Sistemas Operativos

Problemas frecuentes con el lenguaje C

Grado en Ingeniería Informática

Universidad Carlos III de Madrid