CC3301
Programación de Software de Sistemas
Profesor: Luis Mateu

- Ejemplo de un memory leak y un dangling reference
- Recolección de basuras vs. free
- Typedef, regla de sustitución para typedef
- Estructuras
- Punteros a estructuras
- Estructuras de datos recursivas
- El puntero nulo

El recolector de basuras

- Un recolector de basuras libera automáticamente las variables dinámicas que ya no son alcanzables por el programa
- C y C++ no poseen recolector de basuras
- Java y Python sí poseen recolector de basuras
  - No hay manera de liberar explícitamente la memoria
  - La ventaja es que no pueden haber errores asociados a *dangling references*
  - Casi no hay *memory leaks*
  - No se pierde tiempo de desarrollo en descubrir en donde se debe liberar la memoria
  - Pero el recolector de basuras introduce un sobrecosto importante en tiempo de ejecución y uso de memoria
  - Las implementaciones más eficientes introducen pausas en la ejecución que son molestas en aplicaciones interactivas
  - Hay recolectores de basuras que minimizan las pausas pero con un sobrecosto aún mayor en tiempo de ejecución

Definición de alias para tipos: Typedef

- La sintaxis: ***T*** *id1, *id2, …* declara que *id1*, *id2*, … son variables que almacenan valores del tipo *T, T*, …*
- Considere que a esta declaración se antepone typedef:

  typedef *T* *id1, *id2, …*

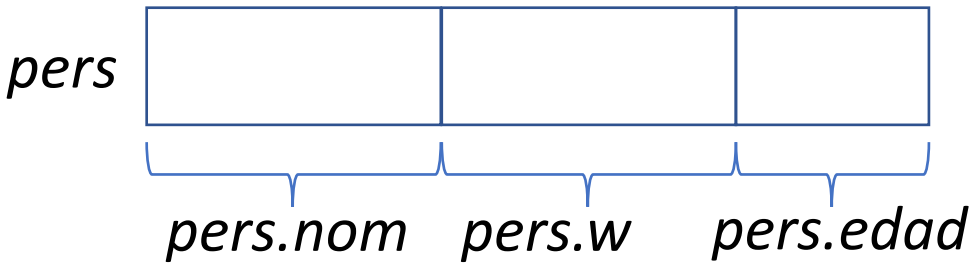
- Ahora los identificadores *id1*, *id2*, … ya no son variables: *¡Son alias tipos!*
- Por ejemplo: typedef int Ent, *P_Ent;
- Ent y P_Ent son tipos
- ¿Cuáles son los tipos concretos de *x, y, z?*

  Ent x, y[10]; P_Ent z;

- *Regla de substitución para typedef*: para saber el tipo de y[10] substituya Ent en el typedef por y[10] y suprima el typedef y los otros tipos declarados
- Queda: int y[10]; Ent es un alias de int
- Por lo tanto y es un arreglo de 10 enteros
- P_Ent es un alias de int*

Estructuras

- Las estructuras son variables de tipo compuesto: almacenan múltiples valores a la vez
- Se declaran con ***struct etiqueta { decl1 decl2 … } …***
- Ej.: `struct persona { char *nom; double w; int edad; } ;`
  `struct persona pers;`
- La variable *pers* es una estructura de tipo *struct persona* que almacena 3 valores de tipos distintos
- *nom w edad* son los campos (fields) de la estructura
- Para seleccionar cada uno de los campos de *pers* se usa esta sintaxis: *pers.nom pers.w pers.edad*
- Son variable del tipo declarado en struct …



- Con *pers.nom* se puede hacer todo lo que se puede hacer con una variable de tipo *char** como *&pers.nom*

Estructuras y typedef

- Como usar *struct persona* es muy largo conviene usar typedef para crear un alias:

```
typedef struct persona Persona;

Persona pedro, ana;  // tipo: struct persona
```

- También se puede usar directamente en la definición de struct persona:

```
typedef struct persona {
    char *nom;
    double w;
    int edad;
}  Persona;
```

```
struct persona pedro; // o
Persona ana;
```

- Si nunca se va a usar *persona* otra vez se puede omitir:

```
typedef struct {
    char *nom;
    double w;
    int edad;
}  Persona;
```

La convención usual es que la primera letra de un alias correspondiente a un struct es mayúscula

Ejemplo: números complejos

- Tipo: typedef struct { double r, im; } Complejo;
- Función que suma números complejos:

```
Complejo suma(Complejo zx, Complejo zy) {
   Complejo res;
   res.r= zx.r + zy.r;
   res.im= zx.im+zy.im;
   return res;
}
```

Las estructuras se pueden retornar y pasar como parámetros

- O más corto:

```
Complejo suma(Complejo zx, Complejo zy) {
   Complejo res = { zx.r + zy.r, zx.im + zy.im };
   return res;
}
```

En la declaración se usa { … } para inicializar los campos de una estructura

- Uso:

```
Complejo a = { 1.3, -10 },  b = { -0.03, 0 };
Complejo c = suma(a, b);
```

- Las estructuras se pueden declarar con inicialización, asignar, pasar como parámetros a una función y ser retornadas, tienen sizeof y dirección

Enfoque imperativo: punteros a estructuras

- Esto ***no funciona***:

```
void sumar(Complejo zx, Complejo zy) {
   zx.r += zy.r;
   zx.im += zy.im;
}
```

- Uso: Complejo a = { 1.3, -10 }, b = { -0.03, 0 };
  sumar(a, b); // ¡a sigue siendo { 1.3, -10 }!
- ***Porque los parámetros se pasan por valor en C: zx es una copia de a, modificar zx no cambia a***
- Se debe usar punteros:

```
void sumar(Complejo *pz, Complejo zy) {
   (*pz).r += zy.r;        // *pz.r += ...
   (*pz).im += zy.im;      // *(pz.r) += ...
}
```

- Uso: Complejo a = { 1.3, -10 }, b = { -0.03, 1 };
  sumar(&a, b); // a es { 1.27, -9 }

Sabor sintáctico

- La expresión *(*p).campo* es tan necesitada que existe un sabor sintáctico equivalente más liviano y legible:

  p -> campo

- Reescritura legible de la función sumar:

```
void sumar(Complejo *pz, Complejo zy) {
   pz->r += zy.r;
   pz->im += zy.im;
}
```

- Si *p* es un puntero a una estructura, acceda a sus campos con *p->campo*
- Si *e* es una estructura, acceda a sus campos con *e.campo*
- La sintaxis general es:

  expresión -> identificador
  expresión . Identificador

```
suma(a, b) . r          // Correcto
~~zy . "r"~~            // Incorrecto
```

Estructuras de datos recursivas

- Estructuras de tipo *T* cuyos campos referencian variables del mismo tipo *T* son *recursivas*
- Ejemplo: un nodo de una lista simplemente enlazada es recursivo porque uno de sus campos es un puntero a otros nodos de la lista
- Declaración:

```
typedef struct nodo {
    char *str;
    struct nodo *prox;
} Nodo;
```

¡No es opcional!

- Forma incorrecta:

```
typedef struct {
    char *str;
    Nodo *prox;  // Error tipo Nodo no existe aún
} Nodo;
```

- *El alias Nodo se hace visible después del typedef*

El puntero nulo

- Función que busca el string *pal* en la lista enlazada cuyo primer nodo es apuntado por *cabeza*:

```
int buscar(Nodo *cabeza, char *pal) {
    while (cabeza!=NULL) {
        if (strcmp(cabeza->str, pal)==0)
            return 1;
        cabeza= cabeza->prox;
    }
    return 0;
}
```

- NULL es la dirección 0
- Si *cabeza* es NULL, *cabeza->str* gatilla *segmentation fault*
- Uso: int presente= buscar(L, "casa");
- ¡L no cambia! Porque el paso de parámetros es por valor