CC41B Auxilar 01

Profesor: L. Mateu
Auxiliar: J.Bustos

1. Diagnóstico C

1.1. Elección Aleatoria

El siguiente programa permite elegir uno de dos enteros pseudo-aleatoriamente.

```
int randomChoose(int a, int b){
    if(time(0)%2) return a;
    return b;
}

main(){
    int a=1,b=2;
    int c=randomChoose(a,b);
}
```

Modifique este programa para que acepte cualquier estructura de C.

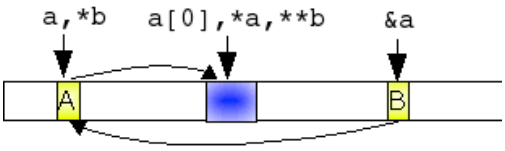
1.2. Swap

El siguiente código para hacer swap no funciona. Explique por que razón, e implemente una versión correcta.

```
void swapMalo(int a, int b){
    int tmp=a;
    a=b;
    b=tmp;
}
```

2. Repaso C

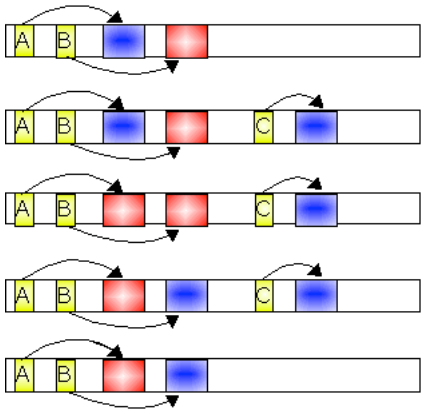
2.1. Punteros



Sopa de Punteros

La siguiente figura muestra los estados de punteros al utilizar una función de swap:

Swap Contenido



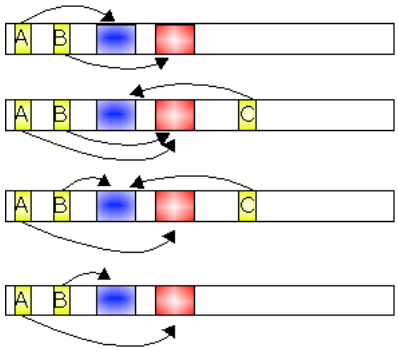
Un intento fallido de implementar este swap resultó en:

```
void swapGenericoMalo(void *a, void *b){
    void *tmp=a;
    a=b;
    b=tmp;
}
```

¿Por qué razón no funciona? Corrija este swap genérico, de tal manera que el método de swap opere con cualquier estructura de C.

Comente porque no es posible implementar una función que realice swap de referencias. ¿Que alternativa propone?

Swap Referencias



2.2. Estructuras

Las estructuras en C puede ser definidas de varias maneras. La tradicional consiste en crear una estructura y definirla como un ti-

po:

```
typedef struct Coord{
    int x,y,z;
} Coord;
```

Para utilizar estas estructuras se debe crear una variable asociada. Por ejemplo, para realizar un swap de estas variables se debe:

```
Coord coordA,coordB;
swapGenerico((void *)&coordA,
    (void *)&coordB, sizeof(Coord));
```

Escriba la llamada a la función de swap genérico, utilizando las siguientes declaraciones de variables:

```
Coord *coordC=
   (Coord *)malloc(sizeof(Coord));
Coord coordD[1];
```

3. Uso de Makefiles

Un `Makefile` es un archivo que permite ejecutar alguna o mas acciones siempre y cuando se cumplan las condiciones lógicas definidas para esa acción.

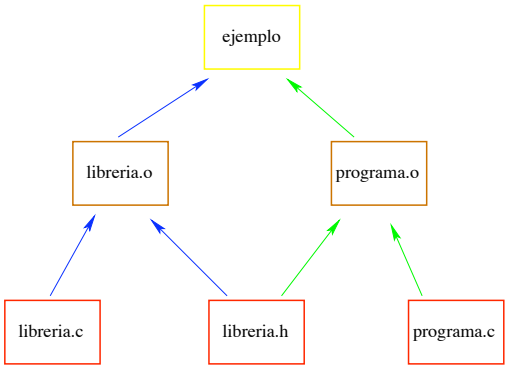
En general los `Makefiles` se utilizan para compilar archivos, como veremos a continuación, pero su uso no está limitado a esta funcionalidad.

Un archivo `Makefile` se compone principalmente de reglas. Cada regla tiene ninguna o mas condiciones (dependecias), y cero o mas instrucciones. Para poder ejecutar las instrucciones de una regla, primero deben satisfacerse las condiciones de la misma.

Veamos, por ejemplo el siguiente caso. Tenemos un `programa.c` que utiliza una `libreria.c` mediante el archivo `libreria.h`. La forma de compilar esto seria:

```
gcc -c libreria.c
gcc -c programa.c
gcc libreria.o programa.o -o ejemplo
```

Gráficamente, lo que estamos haciendo es crear un árbol donde los nodos padre dependen de la existencia de los hijos antes de poder ser compilados.



Las dependencias anteriores, junto con las acciones de compilar, pueden ser reguladas mediante un `Makefile` como el que se muestra a continuación.

Cada linea del archivo que contiene una palabra, seguida de un `:` es considerada una nueva regla. Sus condiciones (dependencias) son las palabras que se encuentran despues de los dos puntos. Estas palabras pueden representar otras reglas del mismo `Makefile` o archivos que deben existir en la misma carpeta. Las lineas siguientes contendrán las instrucciones que deben ser ejecutadas al satisfacer esa regla.

Aqui la regla defecto se satisface si y sólo si se cumple alguna de las siguientes condiciones:

- Existe alguna regla llamada `ejemplo`, y esta se satisface recursivamente.
- Existe un archivo llamado ejemplo y su fecha de modifiación es mas reciente que todas sus dependencias, si es que existe alguna.

Cuando alguna relga del archivo sea satisfecha, se ejecutarán sus respectivas instrucciones.

Para mayor información respecto a los `Makefiles` pueden visitar el siguiente tutorial en Internet:
http://www.eng.hawaii.edu/Tutor/Make/.

4. Soluciones

4.1. Diagónstico

```
int randomChoose(int a, int b){

    if(time(0)%2)
        return a;

    return b;
}

void * genericRandomChoose(void *a,
                           void *b){

    if(time(0)%2)
        return a;
    else
        return b;
}

main(){

    int a=1;
    int b=2;

    /* Alternativas
    void *c=genericRandomChoose(&a,&b);
    printf("selected:%d\n", *(int *)c);

    int *c=
        (int *)genericRandomChoose(&a,&b);
    printf("selected:%d\n", *c);

    */

    int c=
       *(int *)genericRandomChoose(&a,&b);
    printf("selected:%d\n", c);
}
```

4.2. Problemas

```
#define swapReferencias(x, y)  { int tmp = x; x = y; y = tmp }

void swapMalo(int a, int b){
    int tmp=a; a=b; b=tmp;
}

void swapInt(int *a, int *b){
    int tmp=*a; *a=*b; *b=tmp;
}

void swapGenericoMalo(void *a, void *b){
    void *tmp=a;
    a=b;
    b=tmp;
}

void * my_memcpy(void *dest, void*src, int size){

    char *dest_char= (char *)dest;
    char *src_char= (char *)src;

    while(size--)
        *dest_char++=*src_char++;

    return dest;
}

void swapGenerico(void *a, void *b, int size){

    char tmp[size];
    //tmp[0]=*(char *)a;
    //tmp[1]=*(char *)(a+1);
    //...
    //tmp[size-1]=*(char *)(a+size-1);

    //mejor utilizar memcpy
    my_memcpy(tmp,a,size);
    my_memcpy(a,b,size);
    my_memcpy(b,tmp, size);
}

typedef struct Coord{
    int x,y,z;
} Coord;

main(){
    int a=a,  b=b;
    printf("%c %c\n",a,b);
    swapInt(&a,&b);
    printf("%c %c\n",a,b);

    Coord coordA,coordB;
    swapGenerico((void *)&coordA,
        (void *)&coordB, sizeof(Coord));

    Coord *coordC=
        (Coord *)malloc(sizeof(Coord));
    Coord coordD[1];

    swapGenerico((void *)coordC,
        (void *)coordD, sizeof(Coord));
}
```