ESTRUCTURAS DE DATOS

PROBLEMAS EVALUACIÓN CONTINUA

Jose Manuel Martínez de la Insua.

PROBLEMAS DE FUNCIONES

Problema 2.1
Indique el efecto de la siguiente función:

```
void intercambia (int *p, int *q)
{
  int &temp= p;
  p=q;
  q=temp;
}
```

El resultado sería error de compilación en la línea : "int &temp = p;" ya que la función recibe como parámetro dos punteros a entero y en dicha línea trata de asignar un puntero a entero a un entero.

Problema 2.2
Indique el efecto de la siguiente función:

```
void intercambia (int *p, int *q)
{
  int *temp= p;
  p=q;
  q=temp;
}
```

La función recibe como parámetro dos punteros a entero (cada uno apunta a una dirección de memoria diferente) y trata de intercambiar el lugar hacia donde apuntan ... pero ambos punteros no están pasados por referencia, es decir, la función cambia internamente el valor de los punteros, pero al finalizar no queda guardado el cambio (al ser de carácter local). La solución sería :

```
void intercambia (int* &p, int* &q){
    int *temp = p;
    p = q;
    q = temp;
}
```

Problema 2.3
Implemente una función swap puntero que intercambie el valor de dos punteros usando paso por referencia. Muestre un ejemplo de llamada.

```
#include <iostream>
#include <stdlib.h>
#include <vector>

using namespace std;

void intercambia (int * &p,int * &q){
        int *temp = p;
        p=q;
        q=temp;
}



int main()
{
        int a = 5;
        int b = 6;
        int *ap = &a;
        int *bp = &b;

        intercambia(ap,bp);

        return 0;
}
```

Intercambiamos el valor de los punteros pero las variables siguen con sus valores : a= 5, b = 6.

Problema 2.4
Implemente la función del problema anterior sin usar paso por referencia. Deberá usar un paso de parámetros al \estilo C", es decir, por medio de un puntero al valor a modificar. Reescriba el ejemplo de llamada.

```
#include <iostream>
#include <stdlib.h>
#include <vector>

using namespace std;

void intercambia (int ** p,int ** q)
 {
        int *temp = *p;
        *p=*q;
        *q=temp;
}


int main()
{
        int a = 5;
        int b = 6;
        int *ap = &a;
        int *bp = &b;

        intercambia(&ap,&bp);

        return 0;
}
```

Usamos como parámetro de la función un puntero a puntero a entero y pasamos como argumento a la función las direcciones de memoria de los punteros a intercambiar, de este modo se cambian los valores hacia donde apuntan los punteros y no se modifica el valor de las variables: a=5, b=6.

Problema 2.5
Consideremos que en un proyecto, un programador tiene la tarea de crear una función que determine si un valor entero, indicado por un puntero, es par. Crea la siguiente función:

```
bool par(int *p)
{
  return (*p%2==0);
}
```

Considere otra función, realizada por otro programador, que usa la primera:

```
int numero_pares (const int *v, int n)
{
  int res=0;
  for (int i=0;i<n;i++)
    if (par(v+i)) res++;
  return res;
}
```

¿Qué problema se va a encontrar la persona encargada de integrar ambos códigos? Si es necesario, incluya las dos funciones en un fichero fuente e intente compilarlo. ¿Qué error es de esperar?

Error de conversión de tipo de dato, ya que intenta llamar a la función que acepta como parámetro un puntero a entero, pasando como argumento un puntero a entero constante, y ese cambio no es posible (sí podría usarse si fuese el caso contrario).

En la captura podemos apreciar exactamente qué tipo de error se produjo :

```
insua@insua-HP-Pavilion-g6-Notebook-PC:~/Escritorio/ED-1/Ejercicios$ make ej4
g++   ej4.cpp -o exeej4
ej4.cpp: In function ‘int numeros_pares(const int*, int)’:
ej4.cpp:14:11: error: invalid conversion from ‘const int*’ to ‘int*’ [-fpermissi
ve]
   if(par(v+i)) res++;
           ^
ej4.cpp:7:6: note:   initializing argument 1 of ‘bool par(int*)’
 bool par (int* p){
      ^
Makefile:12: fallo en las instrucciones para el objetivo 'ej4'
make: *** [ej4] Error 1
```

Problema 2.6
Considere la siguiente función:

```
void copiar (float **m, int f, int c, float **& res)
{
  // modifica res, para apuntar a una nueva zona y
  // copia la matriz fxc m a res
}
```

¿Qué posible problema nos podemos encontrar al ejecutar Copiar(mat,f,c,mat)?
¿Cómo podríamos asegurarnos de resolver ese problema?

Un posible problema que podríamos encontrarnos es que al modificar "res" para apuntar a una nueva zona de memoria , la zona de memoria a la que apuntemos podría machacar los valores de la matriz "m", entonces nunca llegaría a copiarse por completo (y si lo hiciese almacenaría basura en algunas posiciones).

*Para asegurarnos de que no nos sucede ese problema el puntero a puntero a float "res" debe apuntar a la dirección de memoria de del puntero a puntero a float "m" + " f * c " posiciones de memoria. (Cada posición de memoria corresponde a 4 Bytes = 32 bits).*

Un ejemplo claro sería:

***m** apunta a la dirección 0x7ffdfdff5170*

***f** = 5*
***c** = 4*

***res** debería apuntar como mínimo a la dirección 0x7ffdfdff51c0.*

*Con 0x7ffdfdff51c0 – 0x7ffdfdff5170 = 80_{10} = f * c * 4 (Bytes).*

Problema 2.7
Las cadenas de caracteres representan un ejemplo clasico en el uso de punteros. El tipo correspondiente para almacenarlas es un vector de caracteres. Implemente las siguientes funciones:

- Función copia cadena. Copia una cadena de caracteres en otra.

```
void copia_cadena ( char * src, char * &dest) {
        dest = src;
        return;
}
```

- Función encadenar cadena. Añade una cadena de caracteres al final de otra.

```
void encadenar_cadena (char * src1, char * src2, char *
&dest) {
                int tam = 0;

                char * m_src1 = src1;
                while (*m_src1 != '\0') {
                        ++m_src1;
                        tam++;
                }
                m_src1 = src1;

                char * m_src2 = src2;
                while (*m_src2 != '\0') {
                        ++m_src2;
                        tam++;
                }
                 m_src2 = src2;

                dest = (char*) malloc( tam*sizeof(char) );

                int index = 0;
                while ( *m_src1 != '\0' )
                        *( dest + index++ ) = *m_src1++ ;
                while ( *m_src2 != '\0' )
                        *( dest + index++ ) = *m_src2++ ;
                return;
        }
```

- Función longitud cadena. Devuelve un entero con la longitud (numero de caracteres sin contar el nulo) de la cadena.

```
int longitud_cadena (const char * str)
{
	int result = 0;
	const char * index = str;

	while( *index != '\0' ) {
		++index;
		result++;
	}

	return result;
}
```

- Función comparar cadena. Compara dos cadenas. Devuelve un valor negativo si la primera es más pequeña", positivo si es más grande" y cero si son \iguales".

```
int comparar_cadena (const char * str1, const char * str2)
{
	int result = 0;
	const char *p1 = str1;
	const char *p2 = str2;
	if(*p1 != '\0' && *p2 != '\0'){
		while (*p1 != '\0' && *p2 != '\0'){
			++p1;
			++p2;
		}
		if (*p1 != '\0' && *p2 == '\0')		result--;
		else if(*p1 == '\0' && *p2 != '\0')	result++;
	}
	else if (*p1 != '\0' && *p2 == '\0')		result--;
	else if(*p1 == '\0' && *p2 != '\0')	result++;

	return result;
}
```

Problema 2.8
En la tabla 1 se muestran posibles pasos de parámetros. Indique si son correctos y por qué.

	Declaración	Parámetro Actual	Parámetro formal	
1	int v	v	float v	✓
2	int m[]	m	int *mat	✗
3	float mat[5]	mat	float *& mat	✗
4	const int v[10]	v	int *mat	✗
5	int m[]	m	int mat[10]	✗
6	int m[3][5][7]	m	int mat[][5][7]	✓
7	float v[5]	v+2	const float mat[]	✓
8	int m[]	m	int mat[][5]	✗
9	float f	f	double f	✓
10	float f	&f	double& f	✗
11	bool mat[5][7]	&mat[3][2]	const bool mat[]	✓
12	char mat[3][5]	mat[0]	char *mat	✓
13	int *m[10]	m	int **mat	✓
14	const double v	&v	double v[]	✗
15	int **m	m	int mat[][]	✗
16	int mat[5][7][9][11]	&mat[0][0][0][0]	int *p	✓
17	int *p	&p	int *mat[]	✓
18	float *p	p	float *& p	✓
19	int m[3][5][7]	m	int *mat[5][7]	✗
20	int mat[5][7][9][11]	mat[2][4][3]	const int *m	✓
21	float *p	p+5	float *& p	✗
22	double *p	p+2	float *p	✗
23	int m[5][10]	m	int *mat[]	✗
24	int *mat[5]	&mat[5]	const int **p	✗
25	const bool *mat	mat+4	bool *p	✗
26		5	int& val	✗

2- No se conoce el tamaño de m.
3 – No se pasan por referencia los vectores y matrices.
4 – No se puede convertir un puntero constante en no constante.
5 – No se conoce el tamaño de m.
8 - No se conoce el tamaño de m.
10 – La función admite un valor double no constante y recibe un puntero a float.
14 – No se puede convertir un puntero constante en no constante.
15 – Se desconoce el tamaño de las columnas.

19 – No se puede convertir un array tridimensional en un puntero a array.
21 - Asume la suma como constante y el parámetro formal es variable.
22 - No se puede convertir puntero a double en puntero a float.
23 – La función tiene como parámetro puntero a puntero y recibe puntero a array.
24 – No se puede convertir un puntero a puntero no constante en un puntero a puntero constante.
25 – No se puede convertir un puntero a bool constante en un puntero a bool no constante.
26 – Asume el número 5 como constante y el parámetro formal es variable.

Problema 2.9
Considere las siguientes funciones:

```
int& primera ()
{ int local; ... return local; }
int& segunda ()
{ static int local=0; ... return local; }
```

¿Cree que son correctas? Razone la respuesta.

La primera es incorrecta y la segunda no. La razón es que en la segunda función se retorna una referencia entera a un valor entero estático (inmutable, luego no se pierde con la ejecución de la función), mientras que en la primera se retorna una referencia a un valor de carácter local, es decir, un valor que se pierde al finalizar la función.

```
insua@insua-HP-Pavilion-g6-Notebook-PC:~/Escritorio/ED-1/Ejercicios$ make
g++   ej1.cpp -o exeej1
ej1.cpp: In function ‘int& primera()’:
ej1.cpp:6:6: warning: reference to local variable ‘local’ returned [-Wreturn-loc
al-addr]
  int local = 0;
      ^
insua@insua-HP-Pavilion-g6-Notebook-PC:~/Escritorio/ED-1/Ejercicios$ ./exeej1
0
Violación de segmento (`core' generado)
```

Problema 2.10
Suponga que ejecutamos el siguiente programa:

```
#include <iostream>
using namespace std;

struct Par{
   int *p1,*p2;
};
void intercambia(Par p)
{
   int *paux;
   paux=p.p1; p.p1=p.p2; p.p2=paux;

   int aux;
   aux=*p.p1; *p.p1=*p.p2; *p.p2=aux;
}

int main()
{
   Par p;
   int x=0,y=1;
   p.p1=&x;
   p.p2=&y;
   intercambia(p);
   if (p.p1==&x)
      cout << ''Los punteros están igual''<<endl;
   if (*p.p1==0)
      cout << ''El primero sigue teniendo 0''<<endl;
}
```

¿Cual cree que sera la salida? Razone la respuesta.

*Los punteros permanecen iguales y los valores de las variables cambian debido a que en la linea “aux = *p.p1;*p.p1 = *p.p2;*p.p2 = aux;” se cambian los valores de los punteros a entero , pero no se cambian las direcciones de memoria a donde apuntan.*