

# FCR PL01

Juan José Vázquez Prieto -> U0282978

Martín Feito Rodríguez -> U0286333

```
#include <iostream>
#include <bitset>
using namespace std;
using std::cout; using std::endl;
using std::string; using std::bitset;
extern "C" bool IsValidAssembly(int a, int b, int c);

char controlString(char* cadena,char* cadena2) { //Esta es la primera función que recibe dos cadenas definidas en main()
    if (strcmp(cadena, "2dBkF998Y") == 0) { //Este condicional imprime en la consola si la primera cadena es 2dBkF998Y
        cout << "Acceso Permitido" << endl;
        return true;
    }
    else {
        cout << "Acceso Denegado" << endl; //En caso contrario da como resultado un mensaje de Acceso Denegado y exit.
        return false;
        exit(1);
    }
    if (strlen(cadena2) ≤ 10 && cadena2[0] == cadena2[9]) { //Este condicional comprueba si la longitud de la cadena 2 es ≤ que 10
        cout << "Acceso Permitido" << endl; //Ademas de comprobar si el elemento 0 (osea el primero) de la cadena2 es = al 9(el 10 en realidad).
        return true;
    } //En caso afirmativo imprime Acceso Permitido y no nos echaria del programa.
    else {
        cout << "Hubo un fallo" << endl; //En caso contrario da como resultado un mensaje de Hubo un fallo y exit.
        return false;
        exit(1);
    }
}

int ControlWithMask(bitset<32> bs1,bitset<32> bs2,int decimal) //Esta función de tipo entero recibe dos numeros binarios de 32 bits ademas de un decimal.
{
    if (bs1[5] == 0) { //Este condicional comprueba si el bit 6 o la 5 posicion del numero 1 es igual a 0.
        cout << "Correcto" << endl; //En caso afirmativo imprime un mensaje de correcto en la pantalla.
        return true;
    }
    else {
        cout << "Hubo algún fallo" << endl; //En caso contrario imprime un mensaje de error y llama al exit.
        return false;
        exit(1);
    }
    if (bs1[12] == bs2[10]) { //Este condicional comprueba si la posición 12 o bit 13 del numero 1 es igual al bit 11 o posicion 10 del número 2.
        cout << "Correcto" << endl; //En caso afirmativo imprime un mensaje de correcto en la pantalla.
        return true;
    }
    else {
        cout << "Hubo algún fallo" << endl; //En caso contrario imprime un mensaje de error y llama al exit.
        return false;
        exit(1);
    }
    if (decimal == 3) { //Este condicional comprueba si las posiciones 5,6 y 7 del número 2 convertidas a decimal son igual a 3.
        cout << "Correcto" << endl; //En caso afirmativo imprime un mensaje de correcto en la pantalla.
        return true;
    }
    else {
        cout << "Hubo algún fallo" << endl; //En caso contrario imprime un mensaje de error y llama al exit.
        return false;
        exit(1);
    }
    return 0;
}

int CheckAsmControl(bitset<32> bs3,bitset<32> bs4,bitset<32> bs5) //Esta es la 3 función que recibe tres numeros binarios de 32 bits.
{
    int num180 = 10110100; //Declaramos num180 que es el 180 en binario.
    if ((bs3[0] == bs5[17]) && ((int)(bs3.to_ulong()) > num180)) //Este condicional comprueba si el bit 1 o posicion 0 del número 3 es igual del bit 18 o pos 17
    {
        cout << "Bien" << endl; //En caso afirmativo imprime un mensaje de correcto en la pantalla.
        return 1;
        return true;
    }
    else
    {
        cout << "Mal" << endl; //En caso contrario imprime un mensaje de error y llama al exit.
        return 0;
        return false;
    }
}
```

```

int CheckInlineAsmAccess(bitset<32> bs6, const char* bs) // Esta función de tipo entero recibe un número binario de 32 bits.
{
    if (num6 == 262144) {
        bs = "000000000000100000000000000000";
        if (bs[19] == bs[18] && bs[18] != bs[13]) {
            cout << "Entrada correcta" << endl;
            return true;
        }
        else {
            cout << "Entrada incorrecta" << endl;
            return false;
            exit(1);
        }
    }
    if (num6 == 12288) {
        bs = "000000000000000011000000000000";
        if (bs[19] == bs[18] && bs[18] != bs[13]) {
            cout << "Entrada correcta" << endl;
            return true;
        }
        else {
            cout << "Entrada incorrecta" << endl;
            return false;
            exit(1);
        }
    }
    else {
        if (bs6[14] == bs6[15] && bs6[15] != bs6[20]) { // Este condicional comprueba si el bit 13 del num6 es igual al 14 y si el bit 14 es igual al 19.
            cout << "Entrada correcta" << endl; //En caso afirmativo imprime un mensaje de correcto en la pantalla.
            return true;
        }
        else {
            cout << "Entrada incorrecta" << endl; //En caso contrario imprime un mensaje de error y llama al exit.
            return false;
            exit(1);
        } //Aquí esta __asm que es donde se implementa el ensamblador en linea el cual conseguimos al debuggear la función y acceder al desensamblado.
        //Lo tenemos comentado debido a que no funciona.
    }
}
// /* __asm
// ...
return 0;
}

```

```

string Decimal_Binario(int n) //Esta función convierte los números enteros a números binarios.
{
    unsigned resto;
    unsigned int numBin = 0;
    unsigned int cont = 0;
    while (n > 0) {
        resto = n % 2;
        n = n / 2;
        numBin = numBin + resto * pow(10, cont);
        cont++;
    }
    return numBin;
}

int Binario_Decimal(int x,int decimal) { //Esta función convierte los números binarios a números decimales.
    int exp, digito;
    exp = 0;
    decimal = 0;
    while (((int)(x / 10)) != 0)
    {
        digito = (int)x % 10;
        decimal = decimal + digito * pow(2.0, exp);
        exp++;
        x = (int)(x / 10);
    }
    decimal = decimal + x * pow(2.0, exp);
    return decimal;
}

```

```

int main() //Esta es la parte principal del programa donde se ejecuta todo el código.
{
    char cadena[10]; //Aqui se declaran todas las variables que se usan.
    char cadena2[10];
    unsigned int num1;
    unsigned int num2;
    int num3;
    int num4;
    int num5;
    int num6;
    int decimal;
    const char* bs;
    cout << "Introduce una cadena: "; //Aqui se solicitan todas las variables necesarias.
    cin >> cadena;
    cout << cadena << endl;

    cout << "Introduce otra cadena: ";
    cin >> cadena2;
    cout << cadena2 << endl;

    cout << "Introduce el num1 (entero positivo): ";
    cin >> num1;
    cout << num1 << endl;

    cout << "Introduce el num2 (entero positivo): ";
    cin >> num2;
    cout << num2 << endl;

    cout << "Introduce el num3 (entero): ";
    cin >> num3;
    cout << num3 << endl;

    cout << "Introduce el num4 (entero): ";
    cin >> num4;
    cout << num4 << endl;

    cout << "Introduce el num5 (entero): ";
    cin >> num5;
    cout << num5 << endl;

    cout << "introduce el num6 (entero): ";
    cin >> num6;
    cout << num6 << endl;
}

```

```

bitset<32> bs1(Decimal_Binario(num1)); //Aqui con la libreria bitset se convierten usando la función Decimal_Binario los numeros enteros a binarios de 32 bits.
bitset<32> bs2(Decimal_Binario(num2));
bitset<32> bs3(Decimal_Binario(num3));
bitset<32> bs4(Decimal_Binario(num4));
bitset<32> bs5(Decimal_Binario(num5));
bitset<32> bs6(Decimal_Binario(num6));
char x = bs2[4]+bs2[5]+bs2[6];

Binario_Decimal(x,decimal);
controlString(cadena, cadena2); //Aqui se hace llamada en main a la primera función que usa dos cadenas.
ControlWithMask(bs1,bs2,decimal); //Aqui se hace llamada en main a la segunda función que usa dos num binarios y la conversion de tres bits de num2 a numero decimal.
CheckDownControl(bs3,bs4,bs5); //Aqui se hace llamada en main a la tercera función que usa tres números binarios.
CheckInlineAsmAccess(bs6, num6, bs); //Aqui se hace llamada en main a la cuarta y ultima función que usa un numero binario.

if (controlString != false && ControlWithMask != false && CheckDownControl != false && CheckInlineAsmAccess != false) //Aqui se hace llamada a si las funciones retornaron o no falso.
    //En caso que no retornaran falso se imprime el mensaje de acceso permitido.
{
    cout << "Acceso permitido" << endl;
}
else {
    cout << "Acceso bloqueado" << endl;
}
cout << endl;
return 0;
}

```

Esta es una explicación del código comentado, he dejado dos archivos cpp unos comentado y otro sin comentar que hace más fácil su lectura de código, pero sin la explicación.

### Respuesta a las preguntas propuestas:

- (0,5 puntos) Dirección de la memoria en la que se encuentra la primera cadena que se lee en la primera función indicada en las instrucciones.

La respuesta es 0019FEA0, esto lo sabemos ya que al debuggear el programa poniendo puntos de ruptura para parar el programa para ver las variables y ver donde se ubican. Adjunto una imagen:

- (0,5 puntos) Dirección de la memoria en la que se sitúa el epílogo de la primera función indicada en las instrucciones y el propio código del epílogo, en forma de código máquina y de mnemónicos.

La respuesta creo que es 0040110E debido a que es lo ultimo en desensamblado de la primera función, si el epilogo se refiere a la primera en vez de a la última parte sería 00401020.

```
    }  
004010E2 EB 29        jmp     controlString+0EDh (040110Dh)  
else {  
    cout << "Hubo un fallo" << endl;  
004010E4 68 B0 2B 40 00    push    offset std::endl<char,std::char_traits<char> > (0402BB0h)  
004010E9 68 F4 83 40 00    push    4083F4h  
004010EE 8B 15 A0 80 40 00  mov     edx,dword ptr [_imp_std::cout (04080A0h)]  
004010F4 52              push    edx  
004010F5 E8 36 08 00 00  call    std::operator<<std::char_traits<char> > (0401930h)  
004010FA 83 C4 08          add    esp,8  
004010FD 8B C8          mov     ecx,eax  
004010FF FF 15 80 80 40 00  call    dword ptr [_imp_std::basic_ostream<char,std::char_traits<char> >::operator<< (0408080h)]  
    exit(1);  
00401105 6A 01          push    1  
00401107 FF 15 94 81 40 00  call    dword ptr [_imp_exit (0408194h)] ►|  
}  
}  
0040110D 5D          pop    ebp  
0040110E C3          ret  
--- No hay archivo de origen ---  
0040110F CC          int    3  
--- C:\Users\juan7\Desktop\Teamwork\Teamwork.cpp ---  
  
int ControlWithMask(std::bitset<32> bs1, std::bitset<32> bs2,int decimal)  
{  
    if (decimal == 1) {  
        cout << "Controlando el bit 0 de la másc  
    }  
    else if (decimal == 2) {  
        cout << "Controlando el bit 1 de la másc  
    }  
    else if (decimal == 3) {  
        cout << "Controlando el bit 2 de la másc  
    }  
    else if (decimal == 4) {  
        cout << "Controlando el bit 3 de la másc  
    }  
    else if (decimal == 5) {  
        cout << "Controlando el bit 4 de la másc  
    }  
    else if (decimal == 6) {  
        cout << "Controlando el bit 5 de la másc  
    }  
    else if (decimal == 7) {  
        cout << "Controlando el bit 6 de la másc  
    }  
    else if (decimal == 8) {  
        cout << "Controlando el bit 7 de la másc  
    }  
    else if (decimal == 9) {  
        cout << "Controlando el bit 8 de la másc  
    }  
    else if (decimal == 10) {  
        cout << "Controlando el bit 9 de la másc  
    }  
    else if (decimal == 11) {  
        cout << "Controlando el bit 10 de la másc  
    }  
    else if (decimal == 12) {  
        cout << "Controlando el bit 11 de la másc  
    }  
    else if (decimal == 13) {  
        cout << "Controlando el bit 12 de la másc  
    }  
    else if (decimal == 14) {  
        cout << "Controlando el bit 13 de la másc  
    }  
    else if (decimal == 15) {  
        cout << "Controlando el bit 14 de la másc  
    }  
    else if (decimal == 16) {  
        cout << "Controlando el bit 15 de la másc  
    }  
    else if (decimal == 17) {  
        cout << "Controlando el bit 16 de la másc  
    }  
    else if (decimal == 18) {  
        cout << "Controlando el bit 17 de la másc  
    }  
    else if (decimal == 19) {  
        cout << "Controlando el bit 18 de la másc  
    }  
    else if (decimal == 20) {  
        cout << "Controlando el bit 19 de la másc  
    }  
    else if (decimal == 21) {  
        cout << "Controlando el bit 20 de la másc  
    }  
    else if (decimal == 22) {  
        cout << "Controlando el bit 21 de la másc  
    }  
    else if (decimal == 23) {  
        cout << "Controlando el bit 22 de la másc  
    }  
    else if (decimal == 24) {  
        cout << "Controlando el bit 23 de la másc  
    }  
    else if (decimal == 25) {  
        cout << "Controlando el bit 24 de la másc  
    }  
    else if (decimal == 26) {  
        cout << "Controlando el bit 25 de la másc  
    }  
    else if (decimal == 27) {  
        cout << "Controlando el bit 26 de la másc  
    }  
    else if (decimal == 28) {  
        cout << "Controlando el bit 27 de la másc  
    }  
    else if (decimal == 29) {  
        cout << "Controlando el bit 28 de la másc  
    }  
    else if (decimal == 30) {  
        cout << "Controlando el bit 29 de la másc  
    }  
    else if (decimal == 31) {  
        cout << "Controlando el bit 30 de la másc  
    }  
    else if (decimal == 32) {  
        cout << "Controlando el bit 31 de la másc  
    }  
}
```

The screenshot shows a debugger interface with two main panes. The top pane displays assembly code for a function named `controlString`. The bottom pane shows a memory dump for address `0x00401020`, displaying both ASCII and hex values.

```

void controlString(char* cadena,char* cadena2) {
00401020 55          push    ebp
00401021 8B EC        mov     ebp,esp
00401023 B9 2B C0 40 00  mov    ecx,offset _22C68F67_Teamwork@cpp (040C02Bh)
00401028 E8 D3 3F 00 00  call   __CheckForDebuggerJustMyCode (0405000h)
    if (strcmp(cadena, "2dBkF998Y") == 0) {
0040102D 68 B0 83 40 00  push   4083B0h
00401032 8B 45 08        mov    eax,dword ptr [cadena]
00401035 50          push   eax
00401036 E8 E1 5A 00 00  call   _strcmp (0406B1Ch)
0040103B 83 C4 08        add    esp,8
0040103E 85 C0        test   eax,eax
00401040 75          ?? ??????
00401041 23          ?? ??????
00401042 68          ?? ??????
00401043 B0          ?? ??????
00401044 2B          ?? ??????
00401045 40          ?? ??????
00401046 00          ?? ??????
129 %

```

Dirección:	Columnas:	Salida
0x00401020	Automático	'Team 'Team 'Team 'Team 'Team El su
0x00401036	Automático	'Team 'Team 'Team 'Team 'Team El su
0x0040104C	Automático	'Team 'Team 'Team 'Team 'Team El su
0x00401062	Automático	'Team 'Team 'Team 'Team 'Team El su
0x00401078	Automático	'Team 'Team 'Team 'Team 'Team El su
0x0040108E	Automático	'Team 'Team 'Team 'Team 'Team El su
0x004010A4	Automático	'Team 'Team 'Team 'Team 'Team El su
0x004010RA	Automático	'Team 'Team 'Team 'Team 'Team El su
Variables locales   Memoria 1   Memoria 2   Procesos   Subprocesos   Tareas   Módulos   Inspección 1   Pila de l		

- (0,5 puntos) Código máquina de la primera instrucción de ensamblador introducida en el ensamblador en línea de la cuarta función indicada en las instrucciones.

Adjunto foto de la respuesta que se obtiene mediante el desensamblado:

The screenshot shows a debugger interface with two main panes. The top pane displays assembly code for a function named `CheckInlineAsmAccess`. The bottom pane shows a memory dump for address `0x004012F0`, displaying both ASCII and hex values.

```

int CheckInlineAsmAccess(std::bitset<32> bs6)
{
004012F0 55          push    ebp
004012F1 8B EC        mov     ebp,esp
004012F3 6A FF        push   0FFFFFFFh
004012F5 68 C9 70 40 00  push   4070C9h
004012FA 64 A1 00 00 00 00  mov    eax,dword ptr fs:[00000000h]
00401300 50          push   eax
00401301 64 89 25 00 00 00 00  mov    dword ptr fs:[0],esp
00401308 83 EC 48        sub    esp,48h
0040130B 56          push   esi
0040130C C7 45 EC 00 00 00 00  mov    dword ptr [ebp-14h],0
00401313 B9 2B C0 40 00        mov    ecx,offset _22C68F67_Teamwork@cpp (040C02Bh)
00401318 E8 73 3F 00 00  call   __CheckForDebuggerJustMyCode (0405290h)
    if (bs6[20] == bs6[19] && bs6[20] != bs6[14]) {
0040131D 6A 14        push   14h
0040131F 8D 45 AC        lea    eax,[ebp-54h]
00401322 50          push   eax
00401323 8D 4D 08        lea    ecx,[bs6]
00401326 E8 45 29 00 00  call   std::bitset<32>::operator[] (0403C70h)
0040132B 89 45 E4        mov    dword ptr [ebp-1Ch],eax
0040132E 8B 4D E4        mov    ecx,dword ptr [ebp-1Ch]
00401331 89 4D E0        mov    dword ptr [ebp-20h],ecx
00401334 C7 45 FC 00 00 00 00  mov    dword ptr [ebp-4],0
0040133B 8B 55 EC        mov    edx,dword ptr [ebp-14h]
0040133E 83 CA 01        or    edx,1
00401341 89 55 EC        mov    dword ptr [ebp-14h],edx
00401344 8B 4D E0        mov    ecx,dword ptr [ebp-20h]
00401347 E8 74 29 00 00  call   std::bitset<32>::reference::operator bool (0403CC0h)

```

### Ejemplos de entradas validas e invalidas para las funciones:

Para la 1<sup>a</sup> función una entrada **valida** en cadena 1 sería “**2dBkF998Y**” y **cualquiera distinta** a esta sería **invalida**. La cadena 2 sería diferente ya que sus condiciones es que tenga una longitud igual o menor a 10 y que su posición 2 sea igual a la 9, un ejemplo de una entrada **valida** sería **0000000000** siendo una cadena de longitud igual a 10 y todas sus posiciones iguales y una **no valida** por ejemplo **0111000000000000** ya que su longitud es mayor a 10 y su posición 2 distinta a la 9.

Para la 2<sup>a</sup> función una entrada **valida** sería el **48 en n2 y 0 en n1** ya que cumple todas las condiciones, el bit 6 del primer número sería 0, el bit 13 de n1 seria igual al bit 11 de n2 y los bits 5,6 y 7 de n2 formarían un 3 en decimal (“011”). Una entrada **no valida** sería **200 en n2 y 48 en n1** ya que no cumplirían las condiciones puestas e imprimiría un mensaje de error además de llamar al exit.

Para la 3<sup>a</sup> función una entrada **valida** sería **181 en num3,0 en num4 y 0 en num5** ya que cumple la condición de igualdad de bits y de ser mayor que 180. Una entrada **no valida** sería **50 en num3, 1 en num4 y 0 en num5** ya que no se cumpliría ninguna condición.

Para la última función una entrada **valida** sería **12288 o el 262144** debido a que ambos cumplen las condiciones de igualdad y diferencia de bits, mientras que **0** sería una **entrada invalida** al no cumplirse la diferencia de bits o **266240** ya que cumple la diferencia, pero no la igualdad.

### Reparto de trabajo:

Yo (Martin) me he encargado de la función 3 y 2. Además de arreglar algunos errores del programa.

Yo (Juan) por mi parte me he encargado de la función 1 y 4 además de arreglar errores y resolver los ejercicios de la fase 1.2 y hacer este pdf.

Miguel Nonide Miranda con U0286609 no participo en el trabajo por tanto solo fue realizado por las dos personas ya nombradas.

Entre los dos le hemos dedicado de media cada uno entre 15 y 20 horas a este trabajo.