

**Arquitectura de computadoras**

Practica 1

**Alumno:**

Efrain Adrian Luna Nevarez

Se diseño un test suite de unittest para probar la funcionalidad de todas las funciones, este test suite se puede observar en el archivo main.c, para evaluar cada función se hace uso de esta función:

short **check\_test**(const char \* functionName,

short test\_number,

const void \* current,

const void \* expected,

unsigned short size)

{

short ret = 0;

#ifdef DEBUG\_WINDOWS\_CONSOLE

if (memcmp(current, expected, size) != 0)

{

ret = -1;

}

if (ret == 0)

{

printf("%s test=%d SUCCESS\n", functionName, test\_number);

}

else

{

printf("%s test=%d FAILED\n", functionName, test\_number);

}

#endif

return ret;

}

1. void vfnMemCpy (unsigned char\* bpSource, unsigned char\* bpDest, unsigned short wSize)  
   -Copia los datos del bloque apuntado por bpSource al bloque apuntado por bpDest.

Casos de prueba:

void **test\_vfnMemCpy**(void)

{

unsigned char bpSource[16];

unsigned char bpDest[16];

unsigned short wSize = 0;

unsigned short i = 0;

short ret = 0;

*////////////////// First test copy 16 bytes ///////////////////////*

wSize = 16;

*//fill source pointer*

for (i = 0; i < wSize; i++)

{

\*(bpSource + i) = 'a';

}

**vfnMemCpy**(bpSource, bpDest, wSize);

ret = **check\_test**("vfnMemCpy()", 1, (void \*)bpDest, (void \*)bpSource, sizeof(unsigned char)\*16);

*////////////////// Second test copy 8 bytes ///////////////////////*

wSize = 8;

*//fill source and expected pointer*

for (i = 0; i < wSize; i++)

{

\*(bpSource + i) = 'b';

}

**vfnMemCpy**(bpSource, bpDest, wSize);

ret = **check\_test**("vfnMemCpy()", 2, (void \*)bpDest, (void \*)bpSource, sizeof(unsigned char)\*16);

*////////////////// Second test copy 4 bytes ///////////////////////*

wSize = 4;

*//fill source and expected pointer*

for (i = 0; i < wSize; i++)

{

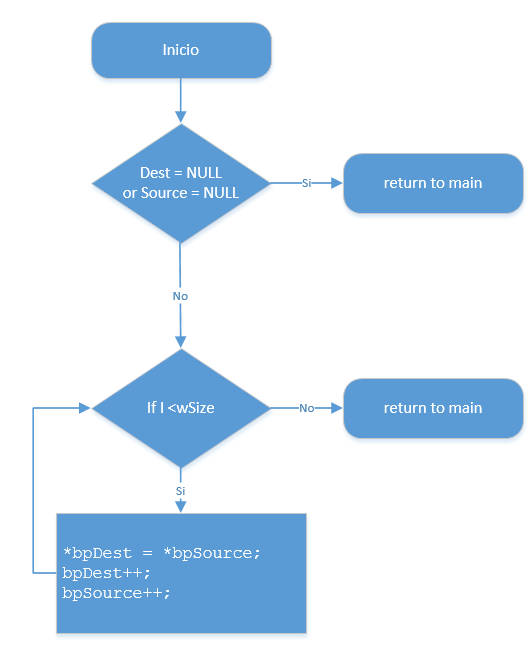
\*(bpSource + i) = 'c';

}

**vfnMemCpy**(bpSource, bpDest, wSize);

ret = **check\_test**("vfnMemCpy()", 3, (void \*)bpDest, (void \*)bpSource, sizeof(unsigned char)\*16);

}



1. void vfnMemSet(unsigned char\* bpDest, unsigned char bByteToFill, unsigned short wSize)  
   -Rellena el bloque apuntado por bpDest con el dato indicado por bByteToFill.

Casos de prueba:

void **test\_vfnMemSet**(void)

{

unsigned char bpDest[16];

unsigned char pExpected[16];

unsigned char bByteToFill = 0;

unsigned short wSize = 0;

unsigned short i = 0;

short ret = 0;

*//////////////// First test fill 16 bytes with 0 ///////////////////*

wSize = 16;

*//fill expected pointer*

for (i = 0; i < wSize; i++)

{

\*(pExpected + i) = bByteToFill;

}

**vfnMemSet**(bpDest, bByteToFill, wSize);

ret = **check\_test**("vfnMemSet()", 1, (void \*)bpDest, (void \*)pExpected, sizeof(unsigned char)\*16);

*//////////////// First test fill 8 bytes with -1 ///////////////////*

wSize = 8;

bByteToFill = -1;

*//fill expected pointer*

for (i = 0; i < wSize; i++)

{

\*(pExpected + i) = bByteToFill;

}

**vfnMemSet**(bpDest, bByteToFill, wSize);

ret = **check\_test**("vfnMemSet()", 2, (void \*)bpDest, (void \*)pExpected, sizeof(unsigned char)\*16);

*//////////////// First test fill 8 bytes with -2 ///////////////////*

wSize = 8;

bByteToFill = -2;

*//fill expected pointer*

for (i = 0; i < wSize; i++)

{

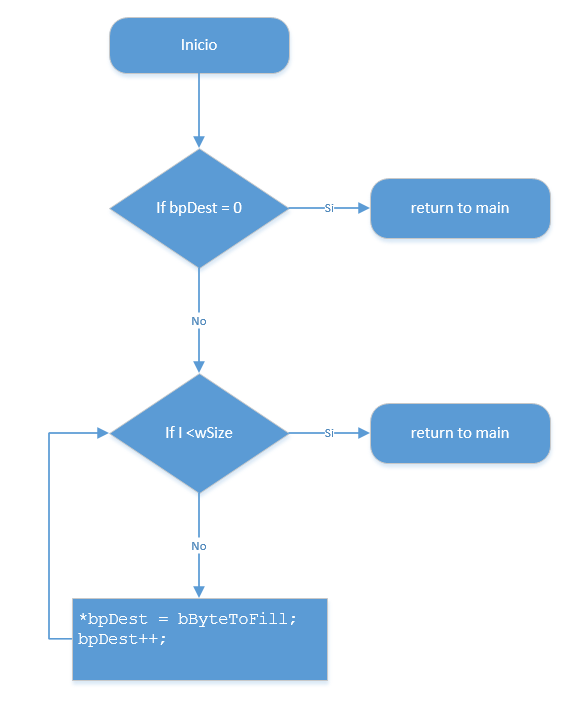
\*(pExpected + i) = bByteToFill;

}

**vfnMemSet**(bpDest, bByteToFill, wSize);

ret = **check\_test**("vfnMemSet()", 3, (void \*)bpDest, (void \*)pExpected, sizeof(unsigned char)\*16);

}



1. unsigned char *bfnFindMax*(unsigned char\* bpDest, unsigned short wSize)  
   -Devuelve el número sin digno mayor encontrado en el bloque de memoria apuntado por bpDest

Casos de prueba:

void **test\_bfnFindMax**(void)

{

unsigned char max = 0;

unsigned char expected = 0;

unsigned char test1[] = {0, 4, 5, 3, 2};

unsigned char test2[] = {10, 15, 5, 7, 12};

unsigned char test3[] = {90, 84, 50, 200, 20};

short ret = 0;

*//////////////// First test ///////////////////*

expected = 5;

max = **bfnFindMax**(test1, 5);

ret = **check\_test**("bfnFindMax()", 1, (void \*)&max, (void \*)&expected, sizeof(unsigned char));

*//////////////// Second test ///////////////////*

expected = 15;

max = **bfnFindMax**(test2, 5);

ret = **check\_test**("bfnFindMax()", 2, (void \*)&max, (void \*)&expected, sizeof(unsigned char));

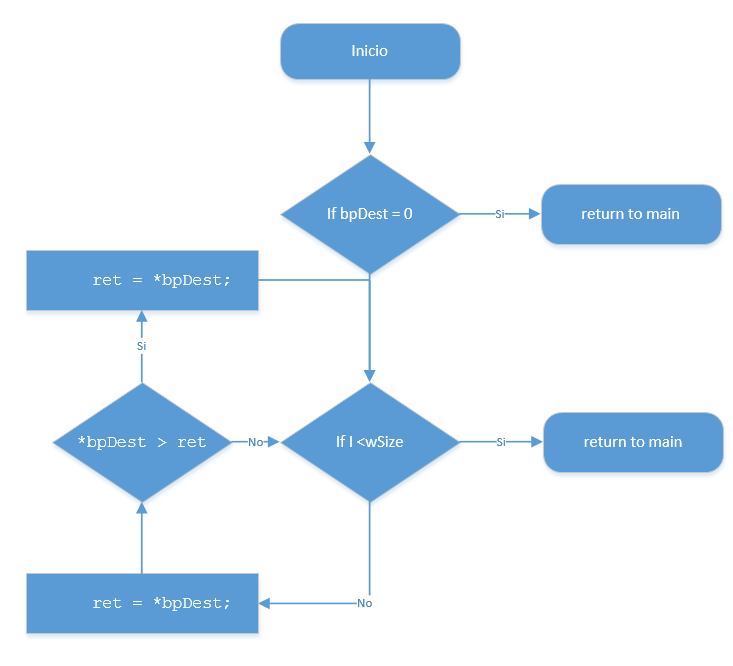
*//////////////// Third test ///////////////////*

expected = 200;

max = **bfnFindMax**(test3, 5);

ret = **check\_test**("bfnFindMax()", 3, (void \*)&max, (void \*)&expected, sizeof(unsigned char));

}



1. unsigned char *bfnFindMin*(unsigned char\* bpDest, unsigned short wSize)  
   -Devuelve el número sin signo menor encontrado en el bloque de memoria apuntado por bpDest

Casos de prueba:

void **test\_bfnFindMin**(void)

{

unsigned char min = 0;

unsigned char expected = 0;

unsigned char test1[] = {0, 4, 5, 3, 2};

unsigned char test2[] = {10, 15, 5, 7, 12};

unsigned char test3[] = {90, 84, 50, 200, 20};

short ret = 0;

*//////////////// First test ///////////////////*

expected = 0;

min = **bfnFindMin**(test1, 5);

ret = **check\_test**("bfnFindMin()", 1, (void \*)&min, (void \*)&expected, sizeof(unsigned char));

*//////////////// Second test ///////////////////*

expected = 5;

min = **bfnFindMin**(test2, 5);

ret = **check\_test**("bfnFindMin()", 2, (void \*)&min, (void \*)&expected, sizeof(unsigned char));

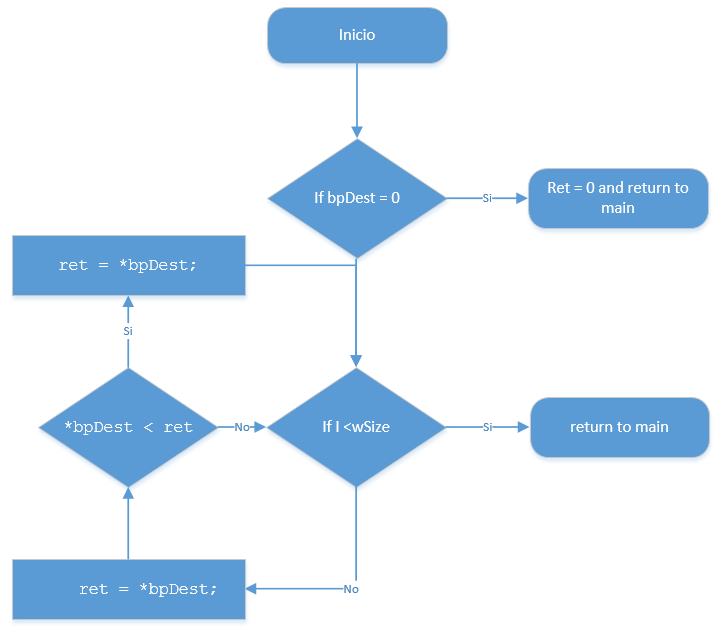
*//////////////// Third test ///////////////////*

expected = 20;

min = **bfnFindMin**(test3, 5);

ret = **check\_test**("bfnFindMin()", 3, (void \*)&min, (void \*)&expected, sizeof(unsigned char));

}



1. unsigned char *bfnStrCmp*(unsigned char\* bpSource, unsigned char\* bpDest, unsigned short wSize)  
   -Compara las cadenas apuntadas por bpSource y bpDest, devolviendo un 1 si las cadenas son iguales o un 0 si son distintas.

Casos de prueba:

void **test\_bfnStrCmp**(void)

{

unsigned char \* test1\_str1 = "Hola mundo";

unsigned char \* test1\_str2 = "Hola mundo";

unsigned char \* test2\_str1 = "Nueva cadena";

unsigned char \* test2\_str2 = "Nueva";

unsigned char \* test3\_str1 = "hola mundo";

unsigned char \* test3\_str2 = "hola Mundo";

unsigned char equal = 0;

unsigned char expected = 0;

short ret = 0;

*//////////////// First test ///////////////////*

expected = 1;

equal = **bfnStrCmp**(test1\_str1, test1\_str2, 10);

ret = **check\_test**("bfnStrCmp()", 1, (void \*)&equal, (void \*)&expected, sizeof(unsigned char));

*//////////////// Second test ///////////////////*

expected = 0;

equal = **bfnStrCmp**(test2\_str1, test2\_str2, 12);

ret = **check\_test**("bfnStrCmp()", 2, (void \*)&equal, (void \*)&expected, sizeof(unsigned char));

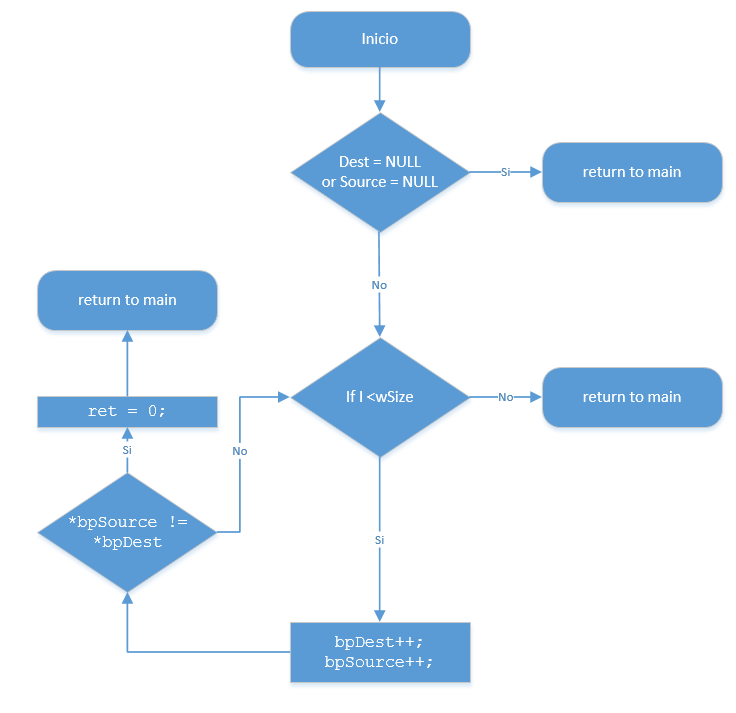
*//////////////// Third test ///////////////////*

expected = 0;

equal = **bfnStrCmp**(test3\_str1, test3\_str2, 10);

ret = **check\_test**("bfnStrCmp()", 3, (void \*)&equal, (void \*)&expected, sizeof(unsigned char));

}



1. unsigned char\* bpfnByteAddress(unsigned char\* bpString, unsigned char bCharToFind, unsigned short wSize)  
   -Entrega la dirección donde se encuentra el valor bCharToFind en la cadena bpString. En caso de no encontrar datos, regresa un apuntador a NULL

Casos de prueba:

void **test\_bpfnByteAddress**(void)

{

unsigned char test[] = {0, 5, 'a', 8, 2};

unsigned char bChar1 = 'a';

unsigned char bChar2 = 'c';

unsigned char bChar3 = 5;

unsigned char\* address = NULL;

unsigned char found = 0;

unsigned char expected = 0;

short ret = 0;

*//////////////// First test ////////////////////*

expected = 1;

address = **bpfnByteAddress**(test, bChar1, 5);

found = (address != NULL) ? 1: 0;

ret = **check\_test**("bpfnByteAddress()", 1, (void \*)&found, (void \*)&expected, sizeof(unsigned char));

*//////////////// Second test ///////////////////*

expected = 0;

address = **bpfnByteAddress**(test, bChar2, 5);

found = (address != NULL) ? 1: 0;

ret = **check\_test**("bpfnByteAddress()", 2, (void \*)&found, (void \*)&expected, sizeof(unsigned char));

*//////////////// Third test ///////////////////*

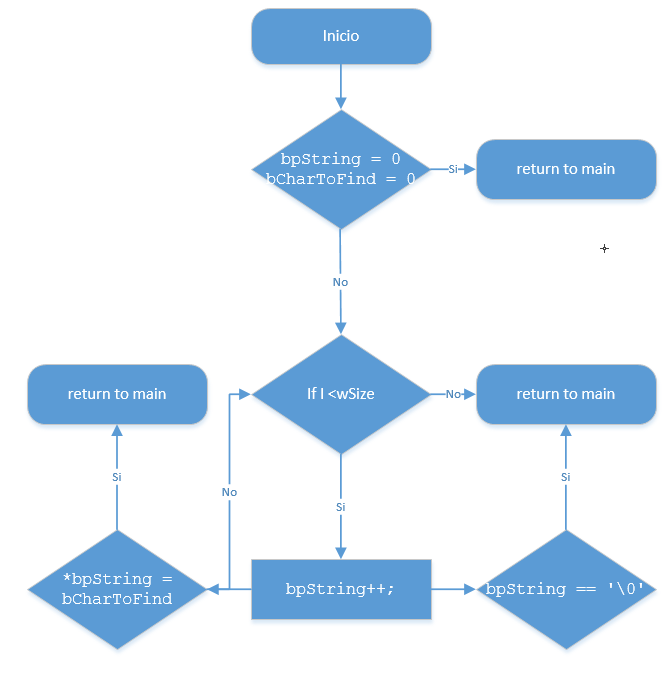
expected = 1;

address = **bpfnByteAddress**(test, bChar3, 5);

found = (address != NULL) ? 1: 0;

ret = **check\_test**("bpfnByteAddress()", 3, (void \*)&found, (void \*)&expected, sizeof(unsigned char));

}



1. unsigned short wfnStrLen (unsigned char\* bpString)  
   -Devuelve el tamaño de la cadena terminada en 0 apuntada por bpString.

Casos de prueba:

void **test\_wfnStrLen**(void)

{

unsigned short res = 0;

unsigned short expected = 0;

short ret = 0;

*//////////////// First test ////////////////////*

expected = 7;

res = **wfnStrLen**("cadena1");

ret = **check\_test**("wfnStrLen()", 1, (void \*)&res, (void \*)&expected, sizeof(unsigned short));

*//////////////// Second test ///////////////////*

expected = 17;

res = **wfnStrLen**("cadena2\_mas\_larga");

ret = **check\_test**("wfnStrLen()", 2, (void \*)&res, (void \*)&expected, sizeof(unsigned short));

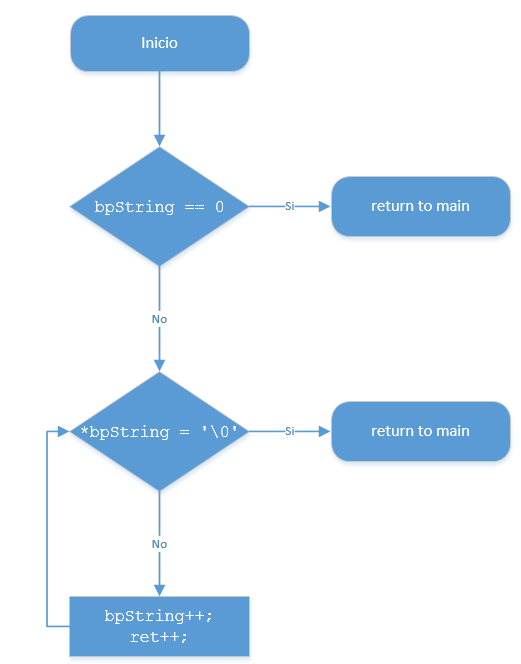
*//////////////// Third test ////////////////////*

expected = 19;

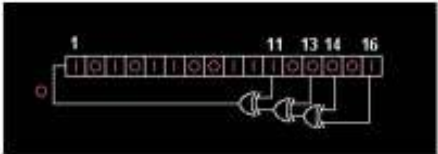
res = **wfnStrLen**("cadena con espacios");

ret = **check\_test**("wfnStrLen()", 3, (void \*)&res, (void \*)&expected, sizeof(unsigned short));

}



1. unsigned short *wfnRand*(unsigned short wSeed)  
   - Devuelve un numero de 16 bits por medio de un generador de números pseudo aleatorio basado en la semilla recibida en wSeed. Utilice el algoritmo de Fibonacci en registros de corrimiento con retroalimentación lineal (LFSR) que muestra la figura:



Casos de prueba:

void **test\_wfnRand**(void)

{

unsigned short res = 0;

short ret = 0;

*// There is not a way to compare the output*

*// so the output for this unit test will always*

*// success*

*//////////////// First test ////////////////////*

res = **wfnRand**(5);

ret = **check\_test**("wfnRand()", 1, (void \*)&res, (void \*)&res, sizeof(unsigned short));

*//////////////// Second test ////////////////////*

res = **wfnRand**(8);

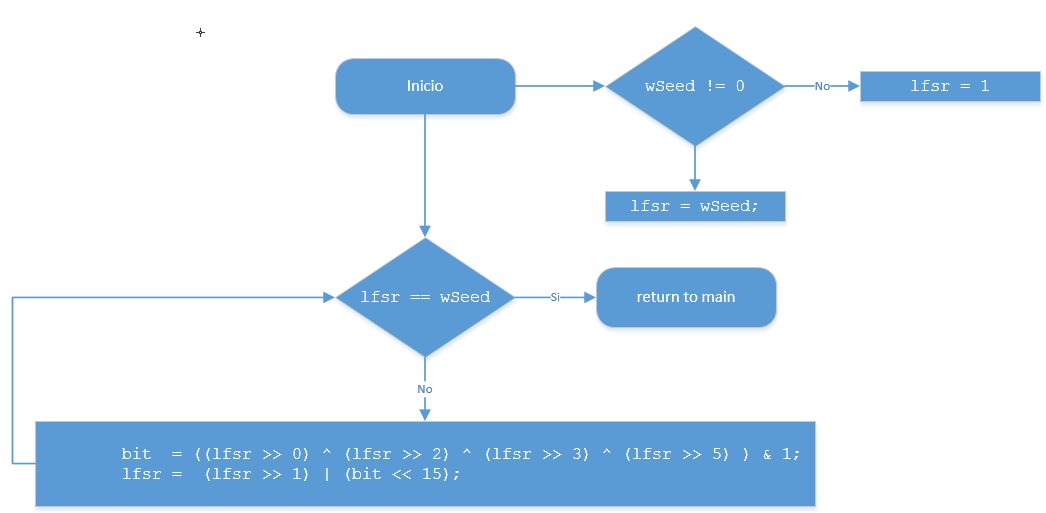
ret = **check\_test**("wfnRand()", 2, (void \*)&res, (void \*)&res, sizeof(unsigned short));

*//////////////// Third test ////////////////////*

res = **wfnRand**(10);

ret = **check\_test**("wfnRand()", 3, (void \*)&res, (void \*)&res, sizeof(unsigned short));

}



1. unsigned char bfnLRC(unsigned char\* bpSrc, unsigned short wSize)  
   - Devuelve el número de 8 bits que resulta de realizar un xor entre todos los datos contenidos en el espacio de memoria apuntado por bpSrc con tamaño especificado por wSize.

Casos de prueba:

void **test\_bfnLRC**(void)

{

unsigned char test1[] = {1, 5, 8, 9, 10};

unsigned char test2[] = {0, 7, 2, 9, 11};

unsigned char test3[] = {0, 0, 1, 0, 1};

unsigned char res = 0;

unsigned char expected = 0;

short ret = 0;

*//////////////// First test ////////////////////*

expected = 15;

res = **bfnLRC**(test1, 5);

ret = **check\_test**("bfnLRC()", 1, (void \*)&res, (void \*)&expected, sizeof(unsigned char));

*//////////////// Second test ////////////////////*

expected = 7;

res = **bfnLRC**(test2, 5);

ret = **check\_test**("bfnLRC()", 2, (void \*)&res, (void \*)&expected, sizeof(unsigned char));

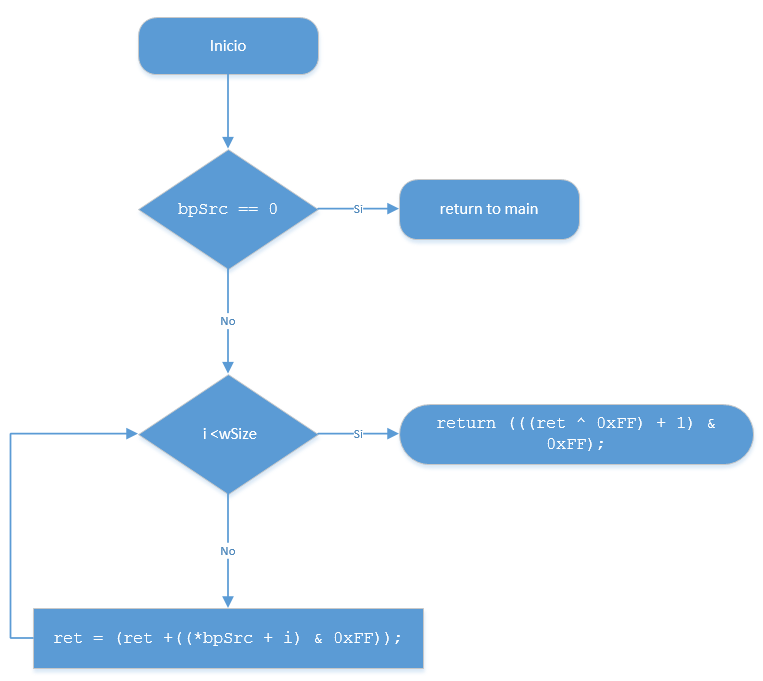
*//////////////// Third test ////////////////////*

expected = 0;

res = **bfnLRC**(test3, 5);

ret = **check\_test**("bfnLRC()", 3, (void \*)&res, (void \*)&expected, sizeof(unsigned char));

}



1. unsigned short *wfnOccurrence*(unsigned char bSymbol, unsigned char\* bpSrc, unsigned short wSize)  
   - Devuelve el número de 16 bits que representa la cantidad de veces que se repite el elemento bSymbol en el bloque indicado por bpSrc de tamaño wSize

Casos de prueba:

void **test\_wfnOccurrence**(void)

{

unsigned char test[] = {'a', 'a', 'a', 'a', 'b', 'c', 'c'};

unsigned short res = 0;

unsigned short expected = 0;

short ret = 0;

*//////////////// First test ////////////////////*

expected = 4;

res = **wfnOccurrence**('a', test, 7);

ret = **check\_test**("wfnOccurrence()", 1, (void \*)&res, (void \*)&expected, sizeof(unsigned short));

*//////////////// Second test ////////////////////*

expected = 1;

res = **wfnOccurrence**('b', test, 7);

ret = **check\_test**("wfnOccurrence()", 2, (void \*)&res, (void \*)&expected, sizeof(unsigned short));

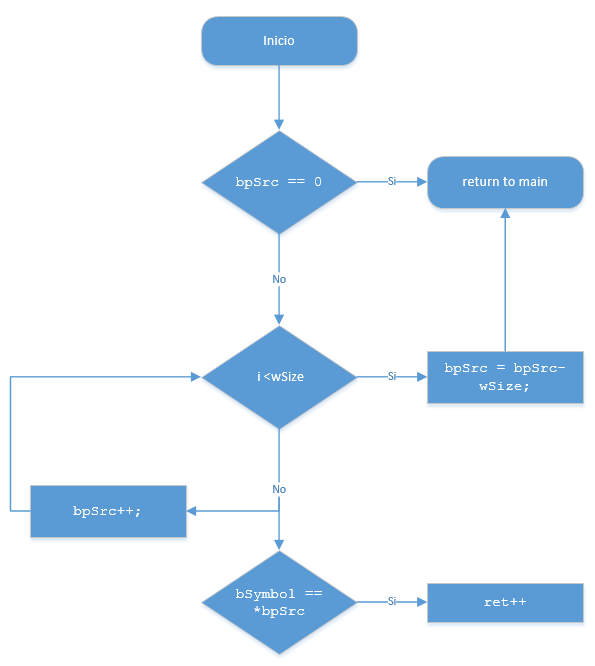
*//////////////// Third test ////////////////////*

expected = 0;

res = **wfnOccurrence**('d', test, 7);

ret = **check\_test**("wfnOccurrence()", 3, (void \*)&res, (void \*)&expected, sizeof(unsigned short));

}



1. void vfnSort (unsigned char\* bpString, unsigned short wStringSize)  
   -Ordena de menor a mayor, en el mismo sitio, los datos contenidos en el buffer apuntado por bpString*.*

Casos de prueba:

void **test\_vfnSort**(void)

{

unsigned char test1[] = {1,0,8,6,4,3,5};

unsigned char test2[] = {8,6,1,0,4,3,5};

unsigned char test3[] = {6,4,3,1,0,8,5};

unsigned char expected[] = {0,1,3,4,5,6,8};

short ret = 0;

*//////////////// First test ////////////////////*

**vfnSort**(test1, 7);

ret = **check\_test**("vfnSort()", 1, (void \*)test1, (void \*)expected, sizeof(unsigned char)\*7);

*//////////////// Second test ////////////////////*

**vfnSort**(test2, 7);

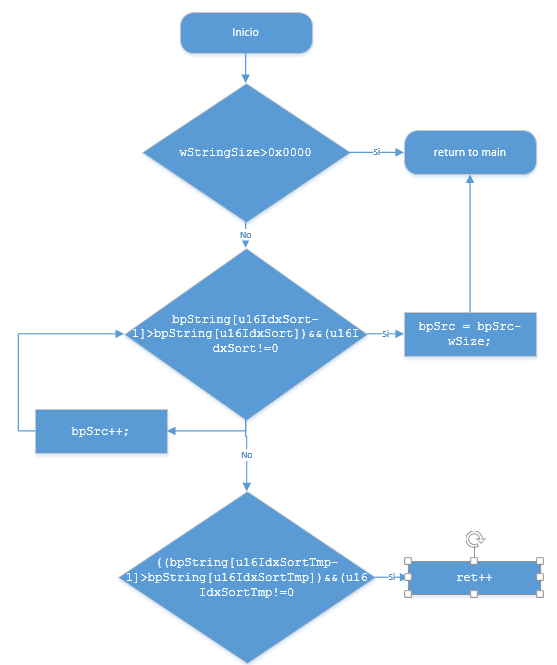
ret = **check\_test**("vfnSort()", 2, (void \*)test2, (void \*)expected, sizeof(unsigned char)\*7);

*//////////////// Third test ////////////////////*

**vfnSort**(test3, 7);

ret = **check\_test**("vfnSort()", 3, (void \*)test3, (void \*)expected, sizeof(unsigned char)\*7);

}



1. unsigned short *wfnSprintf* (char\* string, const char\* fmt, …)  
   -Formatea en la cadena string los datos siguiendo el formato. Devuelve el tamaño de la cadena generada  
   %c – imprime el carácter  
   %d – imprime en 3 dígitos el número de 8 bits  
   %s – imprime como cadena los datos actuales  
   %x – imprime en 2 dígitos el número hexadecimal  
   %b – imprime 8 dígitos representando el binario

Casos de prueba:

void **test\_wfnSprintf**(void)

{

void \* args[5];

unsigned char fmt\_test1[] = "str=%s char=%c int=%d bin=%b hex=%x";

unsigned char fmt\_test2[] = "res=%d hex=%x str=%s";

unsigned char fmt\_test3[] = "number=%0d bin=%0b hex=%0x";

unsigned char exp\_test1[] = "str=cadena char=a int=50 bin=110010 hex=32";

unsigned char exp\_test2[] = "res=50 hex=32 str=cadena";

unsigned char exp\_test3[] = "number=00050 bin=0000000000110010 hex=0032";

unsigned char out\_test1[64];

unsigned char out\_test2[64];

unsigned char out\_test3[64];

unsigned char str[] = "cadena";

unsigned char c = 'a';

unsigned short number = 50;

unsigned short len = 0;

unsigned short exp\_len = 0;

unsigned short i = 0;

short ret = 0;

*//Clean output*

for (i = 0; i < 64; i++)

{

out\_test1[i] = 0;

out\_test2[i] = 0;

out\_test3[i] = 0;

}

*//////////////// First test ////////////////////*

args[0] = &str;

args[1] = &c;

args[2] = &number;

args[3] = &number;

args[4] = &number;

exp\_len = 42;

len = **wfnSprintf**(out\_test1, fmt\_test1, args);

ret = **check\_test**("wfnSprintf()", 1, (void \*)out\_test1, (void \*)exp\_test1, sizeof(unsigned char)\*len);

ret = **check\_test**("wfnSprintf() - len", 2, (void \*)&len, (void \*)&exp\_len, sizeof(unsigned short));

*//////////////// Second test ////////////////////*

args[0] = &number;

args[1] = &number;

args[2] = &str;

args[3] = NULL;

args[4] = NULL;

exp\_len = 24;

len = **wfnSprintf**(out\_test2, fmt\_test2, args);

ret = **check\_test**("wfnSprintf()", 3, (void \*)out\_test2, (void \*)exp\_test2, sizeof(unsigned char)\*len);

ret = **check\_test**("wfnSprintf() - len", 4, (void \*)&len, (void \*)&exp\_len, sizeof(unsigned short));

*//////////////// Third test ////////////////////*

args[0] = &number;

args[1] = &number;

args[2] = &number;

args[3] = NULL;

args[4] = NULL;

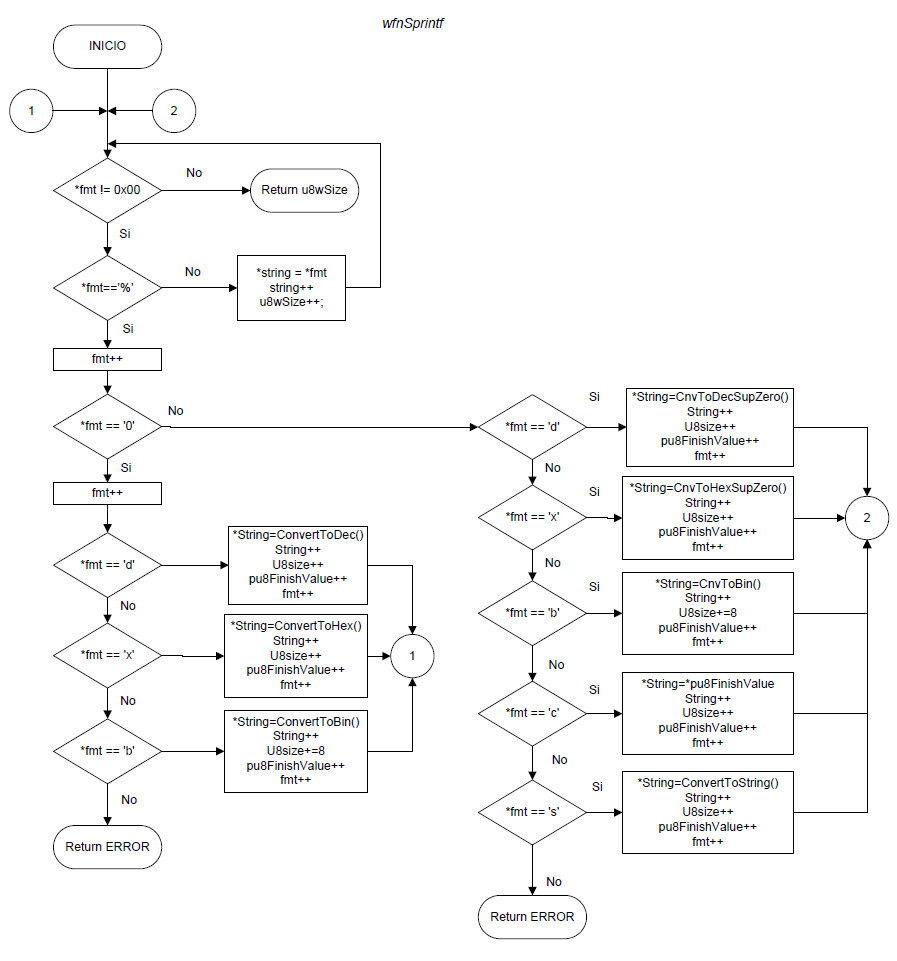
exp\_len = 42;

len = **wfnSprintf**(out\_test3, fmt\_test3, args);

ret = **check\_test**("wfnSprintf()", 5, (void \*)out\_test3, (void \*)exp\_test3, sizeof(unsigned char)\*len);

ret = **check\_test**("wfnSprintf() - len", 6, (void \*)&len, (void \*)&exp\_len, sizeof(unsigned short));

}



**Conclusiones:**

Se aprendió a crear un código de funciones básicas del lenguaje C, las cuales pueden ser portables a diferentes arquitecturas, ya que no son dependientes ni de la arquitectura ni del compilador. Se vio la ventaja de tener un set de pruebas automatizado para realizar las pruebas unitarias de cada función.