

**Arquitectura de computadoras**

Practica 2

**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;

}

**Memory Management:**

Funciones de soporte de manejo dinamico de memoria (Heap)  
• void\* vpfnMalloc(unsigned short wSize) – regresa un apuntador a la posición en memoria reservada para el bloque de tamanio wSize solicitado Si no hay memoria, se regresa un apuntador null.

Casos de prueba:

void **test\_vpfnMalloc**(void)

{

void \* test1\_ptr = NULL;

void \* test2\_ptr = NULL;

void \* test3\_ptr = NULL;

unsigned char was\_allocated = 0;

unsigned char exp\_allocated = 1;

short ret = 0;

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

test1\_ptr = **vpfnMalloc**(sizeof(unsigned short));

was\_allocated = (test1\_ptr != NULL) ? 1:0;

ret = **check\_test**("vpfnMalloc()", 1, (void \*)&was\_allocated, (void \*)&exp\_allocated, sizeof(unsigned char));

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

test2\_ptr = **vpfnMalloc**(sizeof(unsigned int));

was\_allocated = (test2\_ptr != NULL) ? 1:0;

ret = **check\_test**("vpfnMalloc()", 2, (void \*)&was\_allocated, (void \*)&exp\_allocated, sizeof(unsigned char));

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

test3\_ptr = **vpfnMalloc**(sizeof(unsigned char)\*16);

was\_allocated = (test3\_ptr != NULL) ? 1:0;

ret = **check\_test**("vpfnMalloc()", 3, (void \*)&was\_allocated, (void \*)&exp\_allocated, sizeof(unsigned char));

*//Free test memory*

**vfnFree**(test1\_ptr);

**vfnFree**(test2\_ptr);

**vfnFree**(test3\_ptr);

}

• void vfnFree( void\* vpPtr ) – libera el espacio de memoria apuntado por vpPtr

Casos de prueba:

void **test\_vfnFree**(void)

{

void \* ptr1 = NULL;

void \* ptr2 = NULL;

void \* ptr3 = NULL;

extern unsigned short g\_numBytesAllocated;

unsigned short exp\_bytesAllocated = 0;

short ret = 0;

*//Allocate memory*

ptr1 = **vpfnMalloc**(sizeof(unsigned short)\*2);

ptr2 = **vpfnMalloc**(sizeof(unsigned char));

ptr3 = **vpfnMalloc**(sizeof(unsigned char)\*32);

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

exp\_bytesAllocated = 33;

**vfnFree**(ptr1);

ret = **check\_test**("vfnFree()", 1, (void \*)&g\_numBytesAllocated, (void \*)&exp\_bytesAllocated, sizeof(unsigned short));

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

exp\_bytesAllocated = 32;

**vfnFree**(ptr2);

ret = **check\_test**("vfnFree()", 2, (void \*)&g\_numBytesAllocated, (void \*)&exp\_bytesAllocated, sizeof(unsigned short));

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

exp\_bytesAllocated = 0;

**vfnFree**(ptr3);

ret = **check\_test**("vfnFree()", 3, (void \*)&g\_numBytesAllocated, (void \*)&exp\_bytesAllocated, sizeof(unsigned short));

}

• void\* realloc (void\* vpPtr, unsigned short wSize) – Se cambia de posición el buffer apuntado por vpPtr para modificar el tamaño al especificado por wSize Si no hay espacio para el nuevo buffer, se regresa el apuntador inicial.

Casos de prueba:

void **test\_realloc**(void)

{

void \* ptr1 = NULL;

void \* ptr2 = NULL;

void \* ptr3 = NULL;

void \* new\_ptr1 = NULL;

void \* new\_ptr2 = NULL;

void \* new\_ptr3 = NULL;

unsigned char current\_ptr = 0;

unsigned char expected\_ptr = 0;

short ret = 0;

*//Allocate memory*

ptr1 = **vpfnMalloc**(sizeof(unsigned short)\*2);

ptr2 = **vpfnMalloc**(sizeof(unsigned char));

ptr3 = **vpfnMalloc**(sizeof(unsigned char)\*32);

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

*//reallocate 8 bytes*

new\_ptr1 = realloc(ptr1, sizeof(unsigned short)\*4);

current\_ptr = (ptr1 != new\_ptr1) ? 1:0;

expected\_ptr = 1;

ret = **check\_test**("realloc()", 1, (void \*)&current\_ptr, (void \*)&expected\_ptr, sizeof(unsigned char));

**vfnFree**(new\_ptr1);

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

*//reallocate 16 bytes*

new\_ptr2 = realloc(ptr2, sizeof(unsigned char)\*16);

current\_ptr = (ptr2 != new\_ptr2) ? 1:0;

expected\_ptr = 1;

ret = **check\_test**("realloc()", 2, (void \*)&current\_ptr, (void \*)&expected\_ptr, sizeof(unsigned char));

**vfnFree**(new\_ptr2);

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

*//reallocate 1024 bytes (return the same pointer)*

new\_ptr3 = realloc(ptr3, sizeof(unsigned char)\*1024);

current\_ptr = (ptr3 != new\_ptr3) ? 1:0;

expected\_ptr = 0;

ret = **check\_test**("realloc()", 3, (void \*)&current\_ptr, (void \*)&expected\_ptr, sizeof(unsigned char));

**vfnFree**(ptr3);

}

• void\* calloc (unsigned short wSize) – regresa un apuntador de a un buffer lleno de 0  
de tamaño wSize Si no hay espacio para crear el buffer, regresa un apuntador null.

Casos de prueba:

void **test\_calloc**(void)

{

void \* ptr1 = NULL;

void \* ptr2 = NULL;

void \* ptr3 = NULL;

unsigned short exp1[] = {0,0};

unsigned char exp2 = 0;

unsigned char exp3[] = {0,0,0,0,0,0,0,0};

short ret = 0;

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

ptr1 = calloc(sizeof(unsigned short)\*2);

ret = **check\_test**("calloc()", 1, (void \*)ptr1, (void \*)exp1, sizeof(unsigned short)\*2);

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

ptr2 = calloc(sizeof(unsigned char));

ret = **check\_test**("calloc()", 2, (void \*)ptr2, (void \*)&exp2, sizeof(unsigned char));

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

ptr3 = calloc(sizeof(unsigned char)\*8);

ret = **check\_test**("calloc()", 3, (void \*)ptr3, (void \*)exp3, sizeof(unsigned char)\*8);

*//free memory*

**vfnFree**(ptr1);

**vfnFree**(ptr2);

**vfnFree**(ptr3);

}

**Queue Management:**Funciones de manejo de estructuras de datos tipo Queue  
• void vfnQueueInit(void) – Inicializa los apuntadores, contadores, etc de manejo del  
Queue a sus valores iniciales.

Casos de prueba:

void **test\_vfnQueueInit**(void)

{

extern unsigned short g\_numBytesUsedQueue;

extern unsigned char g\_queue[];

extern char \* g\_currentQueueEntry;

unsigned char exp\_queue[256];

unsigned short exp\_numb = 0;

unsigned short i = 0;

short ret = 0;

for (i = 0; i < 256; i++)

{

exp\_queue[i] = 0;

}

**vfnQueueInit**();

*/////////////////////////// Test Values ///////////////////////////*

ret = **check\_test**("vfnQueueInit()", 1, (void \*)g\_queue, (void \*)exp\_queue, sizeof(unsigned char)\*256);

ret = **check\_test**("vfnQueueInit()", 2, (void \*)&g\_numBytesUsedQueue, (void \*)&exp\_numb, sizeof(unsigned short));

ret = **check\_test**("vfnQueueInit()", 3, (void \*)g\_currentQueueEntry, (void \*)g\_queue, sizeof(unsigned char)\*256);

}

• unsigned char bfnEnqueue(char\* cpData, unsigned short wSize) – mete el total de wSize datos apuntados por cpData en el Queue y regresa la cantidad de datos que se pudieron ingresar. Si el queue está lleno, entonces se regresa 0. Si solo se puede meter una parte, entonces se regresa la cantidad que si entro.

Casos de prueba:

void **test\_bfnEnqueue**(void)

{

char data1[16];

char data2[120];

char data3[250];

unsigned short res = 0;

unsigned short exp = 0;

short ret = 0;

*//init queue*

**vfnQueueInit**();

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

exp = 16;

res = **bfnEnqueue**(data1, 16);

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

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

exp = 120;

res = **bfnEnqueue**(data2, 120);

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

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

exp = 120;

res = **bfnEnqueue**(data3, 250);

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

}

• unsigned char bfnDequeue(char\* cpData, unsigned short wSize) – extrae a cpData la cantidad de datos indicadas por wSize del Queue. Se regresa la cantidad de datos extraidos. Si el queue está vacio, se regresa 0.

Casos de prueba:

void **test\_bfnDequeue**(void)

{

char data1[16];

char data2[120];

char data3[250];

char queue\_data[256];

unsigned short res = 0;

unsigned short exp = 0;

short ret = 0;

*//init stack*

**vfnQueueInit**();

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

exp = 0; *//Empty queue*

res = **bfnDequeue**(data1, 16);

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

*//Fill Queue*

**bfnEnqueue**(queue\_data, 256);

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

exp = 120;

res = **bfnDequeue**(data2, 120);

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

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

exp = 136;

res = **bfnDequeue**(data3, 250);

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

}

**Stack Management:**Funciones de manejo de estructuras de datos tipo Stack  
•void vfnStackInit(void) – inicializa los apuntadores, contadores, etc de manejo del  
Stack a sus valores iniciales

Casos de prueba:

void **test\_vfnStackInit**(void)

{

extern unsigned short g\_numBytesUsedStack;

extern unsigned char g\_stack[];

extern char \* g\_currentStackEntry;

unsigned char exp\_stack[256];

unsigned short exp\_numb = 0;

unsigned short i = 0;

short ret = 0;

for (i = 0; i < 256; i++)

{

exp\_stack[i] = 0;

}

**vfnStackInit**();

*/////////////////////////// Test Values ///////////////////////////*

ret = **check\_test**("vfnStackInit()", 1, (void \*)g\_stack, (void \*)exp\_stack, sizeof(unsigned char)\*256);

ret = **check\_test**("vfnStackInit()", 2, (void \*)&g\_numBytesUsedStack, (void \*)&exp\_numb, sizeof(unsigned short));

ret = **check\_test**("vfnStackInit()", 3, (void \*)g\_currentStackEntry, (void \*)g\_stack, sizeof(unsigned char)\*256);

}

• unsigned char bfnStackPush(char\* cpData, unsigned short wSize) – mete el total de wSize datos apuntados por cpData en el Stack y regresa la cantidad de datos que se pudieron ingresar. Si el Stack está lleno, entonces regresa 0. Si solo se puede meter una parte, entonces regresa la cantidad de datos que si entro.

Casos de prueba:

void **test\_bfnStackPush**(void)

{

char data1[16];

char data2[120];

char data3[250];

unsigned short res = 0;

unsigned short exp = 0;

short ret = 0;

*//init stack*

**vfnStackInit**();

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

exp = 16;

res = **bfnStackPush**(data1, 16);

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

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

exp = 120;

res = **bfnStackPush**(data2, 120);

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

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

exp = 120;

res = **bfnStackPush**(data3, 250);

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

}

• unsigned char bfnStackPop(char\* cpData, unsigned short wSize) – extrae a cpData la cantidad de datos indicada por wSize del Stack. Se regresa la cantidad de datos extraidos. Si el Stack está vacio, se regresa 0.

Casos de prueba:

void **test\_bfnStackPop**(void)

{

char data1[16];

char data2[120];

char data3[250];

char stack\_data[256];

unsigned short res = 0;

unsigned short exp = 0;

short ret = 0;

*//init stack*

**vfnStackInit**();

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

exp = 0; *//Empty stack*

res = **bfnStackPop**(data1, 16);

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

*//Fill Stack*

**bfnStackPush**(stack\_data, 256);

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

exp = 120;

res = **bfnStackPop**(data2, 120);

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

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

exp = 136;

res = **bfnStackPop**(data3, 250);

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

}

**General Functions:**Funciones genéricas

• unigned char bfnCmdLine(char\* cpCmd, char\*\* cpCmdList) – Recibe en cpCmdList un apuntador a la lista de comandos válidos y en cpCmd el comando a verificar. Si el comando a verificar se encuentra en la lista de comandos válidos, se regresa el índice correspondiente a la posición de la lista en la que se encontró el comando.

Casos de prueba:

void **test\_bfnCmdLine**(void)

{

char \*cmdList[5];

char cmd[] = "";

char cmd1[] = "command\_1";

char cmd2[] = "command\_2";

char cmd3[] = "command\_3";

char cmd4[] = "command\_4";

char cmd6[] = "command\_6";

unsigned char index = 0;

unsigned char exp\_index = 0;

short ret = 0;

cmdList[0] = cmd;

cmdList[1] = cmd1;

cmdList[2] = cmd2;

cmdList[3] = cmd3;

cmdList[4] = cmd4;

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

exp\_index = 1;

index = **bfnCmdLine**(cmd1, cmdList);

ret = **check\_test**("bfnCmdLine()", 1, (void \*)&index, (void \*)&exp\_index, sizeof(unsigned char));

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

exp\_index = 3;

index = **bfnCmdLine**(cmd3, cmdList);

ret = **check\_test**("bfnCmdLine()", 2, (void \*)&index, (void \*)&exp\_index, sizeof(unsigned char));

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

exp\_index = 0;

index = **bfnCmdLine**(cmd6, cmdList);

ret = **check\_test**("bfnCmdLine()", 3, (void \*)&index, (void \*)&exp\_index, sizeof(unsigned char));

}

• unsigned char bfnLog2 (unsigned long dwNum) – regresa la cantidad mínima de bits necesarios para representar el numero recibido en dwNum

Casos de prueba:

void **test\_bfnLog2**(void)

{

unsigned long dwNum = 0;

unsigned char res = 0;

unsigned char exp = 0;

short ret = 0;

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

exp = 1;

dwNum = 1;

res = **bfnLog2**(dwNum);

ret = **check\_test**("bfnLog2()", 1, (void \*)&res, (void \*)&exp, sizeof(unsigned char));

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

exp = 5;

dwNum = 30;

res = **bfnLog2**(dwNum);

ret = **check\_test**("bfnLog2()", 2, (void \*)&res, (void \*)&exp, sizeof(unsigned char));

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

exp = 7;

dwNum = 120;

res = **bfnLog2**(dwNum);

ret = **check\_test**("bfnLog2()", 3, (void \*)&res, (void \*)&exp, sizeof(unsigned char));

}

**Data Conversion**Funciones de conversión de datos  
• unsigned long dwAToUL (char\* cpPtr) – regresa el numero de 32 bits sin signo que resulta de convertir el numero representado en la cadena terminada en 0 apuntada por cpPtr Si se encuentra un carácter invalido, entonces se regresa 0.

Casos de prueba:

void **test\_dwAToUL**(void)

{

char test1[] = "12345";

char test2[] = "45668678";

char test3[] = "12313a123";

unsigned long res = 0;

unsigned long exp = 0;

short ret = 0;

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

exp = 12345;

res = **dwAToUL**(test1);

ret = **check\_test**("dwAToUL()", 1, (void \*)&res, (void \*)&exp, sizeof(unsigned long));

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

exp = 45668678;

res = **dwAToUL**(test2);

ret = **check\_test**("dwAToUL()", 2, (void \*)&res, (void \*)&exp, sizeof(unsigned long));

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

exp = 0;

res = **dwAToUL**(test3);

ret = **check\_test**("dwAToUL()", 3, (void \*)&res, (void \*)&exp, sizeof(unsigned long));

}

• unsigned short wAToW (char\* cpPtr) – regresa el numero de 16 bits sin signo que resulta de convertir el numero representado en la cadena terminada en 0 apuntada por cpPtr Si se encuentra un carácter invalido, entonces se regresa 0.

Casos de prueba:

void **test\_wAToW**(void)

{

char test1[] = "1234";

char test2[] = "4566";

char test3[] = "342b";

unsigned short res = 0;

unsigned short exp = 0;

short ret = 0;

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

exp = 1234;

res = **wAToW**(test1);

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

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

exp = 4566;

res = **wAToW**(test2);

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

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

exp = 0;

res = **wAToW**(test3);

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

}

• unsigned char bAtoB(char\* cpPtr) – regresa el numero de 8 bits sin signo que resulta de convertir el numero representado en la cadena terminada en 0 apuntada por cpPtr Si se encuentra un carácter invalido, entonces se regresa 0.

Casos de prueba:

void **test\_bAtoB**(void)

{

char test1[] = "123";

char test2[] = "66";

char test3[] = "1a";

unsigned char res = 0;

unsigned char exp = 0;

short ret = 0;

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

exp = 123;

res = **bAtoB**(test1);

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

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

exp = 66;

res = **bAtoB**(test2);

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

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

exp = 0;

res = **bAtoB**(test3);

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

}

• unsigned char bfnBToA(unsigned char bData, char\* cpPtr) – regresa el total de caracteres agregados a la cadena apuntada por cpPtr resultantes de convertir bData a su representación en dígitos ASCII.

Casos de prueba:

void **test\_bfnBToA**(void)

{

unsigned char test1 = 12;

unsigned char test2 = 120;

unsigned char test3 = 200;

char out1[8];

char out2[8];

char out3[8];

unsigned char res = 0;

unsigned char exp = 0;

short ret = 0;

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

exp = 2;

res = **bfnBToA**(test1, out1);

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

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

exp = 3;

res = **bfnBToA**(test2, out2);

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

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

exp = 3;

res = **bfnBToA**(test3, out3);

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

}

• unsigned char bfnWToA(unsigned short wData, char\* cpPtr) – regresa el total de caractere agregados a la cadena apuntada por cpPtr resultantes de convertir wData a su representación en dígitos ASCII.

Casos de prueba:

void **test\_bfnWToA**(void)

{

unsigned short test1 = 1200;

unsigned short test2 = 11220;

unsigned short test3 = 200;

char out1[8];

char out2[8];

char out3[8];

unsigned char res = 0;

unsigned char exp = 0;

short ret = 0;

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

exp = 4;

res = **bfnWToA**(test1, out1);

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

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

exp = 5;

res = **bfnWToA**(test2, out2);

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

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

exp = 3;

res = **bfnWToA**(test3, out3);

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

}

•unsigned char bfnDwToA(unsigned long dwData, char\* cpPtr) – regresa el total de caracteres agregados a la cadena apuntada por cpPtr resultantes de convertir dwData a su representación en dígitos ASCII.

Casos de prueba:

void **test\_bfnDwToA**(void)

{

unsigned long test1 = 14589782;

unsigned long test2 = 1245870;

unsigned long test3 = 245600;

char out1[16];

char out2[16];

char out3[16];

unsigned char res = 0;

unsigned char exp = 0;

short ret = 0;

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

exp = 8;

res = **bfnDwToA**(test1, out1);

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

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

exp = 7;

res = **bfnDwToA**(test2, out2);

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

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

exp = 6;

res = **bfnDwToA**(test3, out3);

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

}

**Data Management**• unsigned char bfnByteFlip(unsigned char bData) – regresa el numero de 8 bits resultante de reflejar cada bit de bData.

Casos de prueba:

void **test\_bfnByteFlip**(void)

{

unsigned char test1 = 0b11110000;

unsigned char test2 = 0b10010000;

unsigned char test3 = 0b11010010;

unsigned char res = 0;

unsigned char exp = 0;

short ret = 0;

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

exp = 0b00001111;

res = **bfnByteFlip**(test1);

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

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

exp = 0b00001001;

res = **bfnByteFlip**(test2);

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

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

exp = 0b01001011;

res = **bfnByteFlip**(test3);

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

}

• unsigned short wfnWordFlip(unsigned short wData) – regresa el numero de 16 bits resultante de reflejar cada bit de wData.

Casos de prueba:

void **test\_wfnWordFlip**(void)

{

unsigned short test1 = 0b1111111100000000;

unsigned short test2 = 0b1001100100000000;

unsigned short test3 = 0b1111110100000010;

unsigned short res = 0;

unsigned short exp = 0;

short ret = 0;

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

exp = 0b0000000011111111;

res = **wfnWordFlip**(test1);

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

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

exp = 0b0000000010011001;

res = **wfnWordFlip**(test2);

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

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

exp = 0b0100000010111111;

res = **wfnWordFlip**(test3);

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

}

• unsigned long dwfnDWordFlip(unsigned long dwData) – regresa el numero de 32 bits resultante de reflejar cada bit de dwData.

Casos de prueba:

void **test\_dwfnDWordFlip**(void)

{

unsigned long test1 = 0xFFFF0000;

unsigned long test2 = 0xFF00FF00;

unsigned long test3 = 0xFF0000FF;

unsigned long res = 0;

unsigned long exp = 0;

short ret = 0;

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

exp = 0x0000FFFF;

res = **dwfnDWordFlip**(test1);

ret = **check\_test**("dwfnDWordFlip()", 1, (void \*)&res, (void \*)&exp, sizeof(unsigned long));

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

exp = 0x00FF00FF;

res = **dwfnDWordFlip**(test2);

ret = **check\_test**("dwfnDWordFlip()", 2, (void \*)&res, (void \*)&exp, sizeof(unsigned long));

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

exp = 0xFF0000FF;

res = **dwfnDWordFlip**(test3);

ret = **check\_test**("dwfnDWordFlip()", 3, (void \*)&res, (void \*)&exp, sizeof(unsigned long));

}

• unsigned char bfnNibbleSwap(unsigned char bData) – regresa el numero de 8 bits resultante de intercambiar los nibbles de bData.

Casos de prueba:

void **test\_bfnNibbleSwap**(void)

{

unsigned char test1 = 0b11110000;

unsigned char test2 = 0b10010010;

unsigned char test3 = 0b11010010;

unsigned char res = 0;

unsigned char exp = 0;

short ret = 0;

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

exp = 0b00001111;

res = **bfnNibbleSwap**(test1);

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

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

exp = 0b00101001;

res = **bfnNibbleSwap**(test2);

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

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

exp = 0b00101101;

res = **bfnNibbleSwap**(test3);

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

}

• unsigned short wfnByteSwap(unsigned short wData) – regresa el numero de 16 bits resultante de intercambiar los bytes de wData.

Casos de prueba:

void **test\_wfnByteSwap**(void)

{

unsigned short test1 = 0b1111111100000000;

unsigned short test2 = 0b1001100100000010;

unsigned short test3 = 0b1111110100001110;

unsigned short res = 0;

unsigned short exp = 0;

short ret = 0;

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

exp = 0b0000000011111111;

res = **wfnByteSwap**(test1);

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

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

exp = 0b0000001010011001;

res = **wfnByteSwap**(test2);

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

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

exp = 0b0000111011111101;

res = **wfnByteSwap**(test3);

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

}

• unsigned long dwfnWordSwap(unsigned long dwData) – regresa el numero de 32 bits resultante de intercambiar los words de dwData

void **test\_dwfnWordSwap**(void)

{

unsigned long test1 = 0xFFFF0000;

unsigned long test2 = 0xFF00FF00;

unsigned long test3 = 0xFF0000FF;

unsigned long res = 0;

unsigned long exp = 0;

short ret = 0;

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

exp = 0x0000FFFF;

res = **dwfnWordSwap**(test1);

ret = **check\_test**("dwfnWordSwap()", 1, (void \*)&res, (void \*)&exp, sizeof(unsigned long));

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

exp = 0xFF00FF00;

res = **dwfnWordSwap**(test2);

ret = **check\_test**("dwfnWordSwap()", 2, (void \*)&res, (void \*)&exp, sizeof(unsigned long));

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

exp = 0x00FFFF00;

res = **dwfnWordSwap**(test3);

ret = **check\_test**("dwfnWordSwap()", 3, (void \*)&res, (void \*)&exp, sizeof(unsigned long));

}

**Comentarios:**

En esta práctica se desarrolló un manejador para el heap y para el stack, los cuales su diseño implico varias horas de desarrollo. Estos modulos pueden ser portados a cualquier microcontrolador y cualquier compilador, ya que no son dependientes ni de la arquitectura ni del compilador. Las otras utilerías resultaron ser mas secillas de implementar, pero son realmente útiles.