**William Collins 040652633**

**Course: CST 8152 – Compilers**

**Lab Section: 11**

**Assignment: 1**

**Professor: Svillen Ranev**

**Due Date: May 31, 2011**

**Date: May 24, 2011**

**Contents:**

Test Plan

buffer.h

buffer.c

ass1fi.out

ass1ai.out

ass1mi.out

ass1e.out

fixed.out

**Test Plan**

The initial tests should match the output files given. Once these files match the ones provided, further tests should be done on larger files to make sure the increment works properly. These tests should test the maximum buffer size.

The final tests will be to alter the main program, calling functions with invalid parameters to make sure the program does not explode.

After a lot of testing, I realized that a short script would be much easier to test with. I have the Cygwin “diff” tool on my windows machine, so this was very easy to do. Here is the batch script I ran:

ass1 ass1.pls fixed > fixed.out

ass1 ass1.pls additive > additive.out

ass1 ass1.pls multiplicative > multi.out

ass1 ass1e.pls fixed > error.out

diff fixed.out ../ass1fi.out

diff additive.out ../ass1ai.out

diff multi.out ../ass1mi.out

diff error.out ../ass1e.out

pause

This was all relative to my directory structure, of course.

**Buffer.h**

/\* File Name: buffer.h

\* Version: 1.11.02

\* Author: William Collins (040652633) && Svillen Ranev

\* Course: CST8152 - Compilers

\* Assignment: 1

\* Date: May 18, 2011

\* Professor: Svillen Ranev

\* Purpose: Declarations and prototypes necessary for the buffer utility.

\*/

#ifndef BUFFER\_H\_

#define BUFFER\_H\_

/\* standard header files \*/

#include <stdio.h> /\* standard input/output \*/

#include <malloc.h> /\* for dynamic memory allocation\*/

#include <limits.h> /\* implementation-defined data type limits \*/

/\* constant definitions \*/

#define R\_FAIL\_1 -1 /\* fail return value \*/

#define R\_FAIL\_2 -2 /\* fail return value \*/

#define LOAD\_FAIL -2 /\* load fail error \*/

#define SET\_R\_FLAG 1 /\* realloc flag set value \*/

#define MULTI\_MIN 1 /\* minimum allowed increment in multiplicative mode \*/

#define MULTI\_MAX 100 /\* maximum allowed increment in multiplicative mode \*/

#define FIXED 0 /\* specifies fixed mode buffer \*/

#define ADDITIVE 1 /\* specifies an additive self-incrementing buffer \*/

#define MULTIPLICATIVE -1 /\* specifies a multiplicative self-incrementing buffer \*/

/\* user data type declarations \*/

typedef struct BufferDescriptor {

char \*ca\_head ; /\* pointer to the beginning of character array (character buffer) \*/

int capacity ; /\* current dynamic memory size (in bytes) allocated to character buffer \*/

char inc\_factor; /\* character array increment factor \*/

int addc\_offset ; /\* the offset (in char elements) to the app-character location \*/

int mark\_offset ; /\* the offset (in chars elements) to the mark location \*/

char r\_flag; /\* reallocation flag \*/

char mode; /\* operational mode indicator\*/

} Buffer ;

/\* function declarations \*/

Buffer \*b\_create(int init\_capacity, char inc\_factor,char o\_mode);

Buffer \*ca\_addc(Buffer \*pB, char symbol);

int b\_reset (Buffer \*pB);

void b\_destroy (Buffer \*pB);

char ca\_isfull (Buffer \*pB);

int ca\_getsize (Buffer \*pB);

int ca\_getcapacity (Buffer \*pB);

int ca\_setmark (Buffer \*pB, int mark);

int ca\_getmark (Buffer \*pB);

int ca\_getmode (Buffer \*pB);

Buffer \*ca\_pack( Buffer \*pB );

int ca\_print (Buffer \*pB);

int ca\_load (FILE \*fi,Buffer \*pB);

#endif

**Buffer.c**

/\* File Name: buffer.c

\* Version: 1.0

\* Author: William Collins (040652633)

\* Course: CST8152 - Compilers

\* Assignment: 1

\* Date: May 18, 2011

\* Professor: Svillen Ranev

\* Purpose: Provide a utility to create and manage a buffer

\* Function list: b\_create, ca\_addc, b\_reset, b\_destroy, ca\_isfull,

ca\_getsize, ca\_getcapacity, ca\_setmark, ca\_getmark,

ca\_getmode, ca\_pack, ca\_print, ca\_load

\*/

#include "buffer.h"

/\*

\* Purpose: Creates a new buffer in memory

\* Author: William Collins (040652633)

\* History/Versions: 1.0

\* Called functions: malloc, free

\* Parameters: init\_capacity - how many chars of storage you want initially

inc\_factor - when storage runs out, this determines how much it will increase by

o\_mode - what mode the buffer is in.

\* Return value: A pointer to the newly created buffer, or NULL on error

\*/

Buffer \*b\_create(int init\_capacity, char inc\_factor, char o\_mode)

{

/\* Allocate memory for the buffer\*/

Buffer \*buff = (Buffer \*)malloc(sizeof(Buffer));

if (buff == NULL)

{

return NULL;

}

/\* Make sure the capacity is a positive amount \*/

if (init\_capacity <= 0)

{

free(buff);

buff = NULL;

return NULL;

}

/\* set the buffer's initial capacity \*/

buff->capacity = init\_capacity;

/\* Allocate memory for the data in the buffer \*/

buff->ca\_head = (char \*) malloc(init\_capacity);

if (buff->ca\_head == NULL)

{

free(buff);

buff = NULL;

return NULL;

}

/\* check for a valid mode, and a valid inc\_factor \*/

if (o\_mode == 'f' || inc\_factor == 0)

{

buff->mode = FIXED;

buff->inc\_factor = 0;

}

else if (o\_mode == 'a')

{

buff->mode = ADDITIVE;

buff->inc\_factor = inc\_factor;

}

else if (o\_mode == 'm' && inc\_factor >= MULTI\_MIN && inc\_factor <= MULTI\_MAX)

{

buff->mode = MULTIPLICATIVE;

buff->inc\_factor = inc\_factor;

}

else

{

/\* This buffer isn't valid, so free up the memory we took and get out of here \*/

free(buff->ca\_head);

free(buff);

buff = NULL;

return NULL;

}

/\* Default the remaining members of the buffer \*/

buff->addc\_offset = 0;

buff->mark\_offset = 0;

buff->r\_flag = 0;

return buff;

}

/\*

\* Purpose: Adds a single character to the buffer, expanding storage if it needs to

\* Author: William Collins (040652633)

\* History/Versions: 1.0

\* Called functions: realloc

\* Parameters: pB - a pointer to the buffer you want to add to

symbol - the character you want to add to the buffer

\* Return value: A pointer to the altered buffer, or NULL on error

\*/

Buffer \*ca\_addc(Buffer \*pB, char symbol)

{

/\*I don't want to work with a null pointer\*/

if (pB == NULL)

{

return NULL;

}

/\* reset the r\_flag to say the last call wasn't to reallocate memory \*/

pB->r\_flag = 0;

/\*If we are about to overflow, get some more memory for the buffer\*/

if (pB->addc\_offset >= pB->capacity)

{

char \*temp; /\* temporary character array that can hold the buffer data while we reallocate \*/

int space\_avail; /\* holds the amount of free space we can realistically use for the buffer data \*/

int new\_inc = 0; /\* holds the increment amount based on buffer mode \*/

switch (pB->mode)

{

case FIXED:

/\* We aren't about to increase a fixed buffer, so get out of here\*/

return NULL;

case ADDITIVE:

/\* Check for int overflow, and assign new capacity to the buffer\*/

if ( pB->capacity + pB->inc\_factor <= 0)

{

return NULL;

}

/\* in additive mode, the increment factor is simply added to the capacity \*/

new\_inc = pB->inc\_factor;

break;

case MULTIPLICATIVE:

/\* Check for overflow, and increase the increase amount if it is okay \*/

if (pB->capacity >= INT\_MAX)

{

return NULL;

}

/\* for multiplicative mode the new increment is calculated using how much space you can reasonably hold \*/

space\_avail = INT\_MAX - pB->capacity;

new\_inc = space\_avail \* (float)pB->inc\_factor / (float)100.00;

break;

}

/\* Reallocate the memory for the data in the buffer\*/

temp = realloc(pB->ca\_head , sizeof(char) \* (pB->capacity + new\_inc));

if (temp == NULL)

{

return NULL;

}

if (pB->ca\_head != temp)

{

pB->r\_flag = SET\_R\_FLAG;

}

pB->ca\_head = temp;

pB->capacity += new\_inc;

}

/\*Now we can finally add the symbol to the buffer\*/

pB->ca\_head[pB->addc\_offset] = symbol;

pB->addc\_offset++;

return pB;

}

/\*

\* Purpose: Resets the data in the buffer

\* Author: William Collins (040652633)

\* History/Versions: 1.0

\* Parameters: pB - pointer to the buffer to be cleared

\* Return value: -1 is returned on error, while 0 is returned on success

\*/

int b\_reset (Buffer \*pB)

{

/\* Can't do anything with a NULL pointer\*/

if (pB == NULL)

{

return R\_FAIL\_1;

}

/\* setting the offset to zero points the array back to the first element \*/

pB->addc\_offset = 0;

return 0;

}

/\*

\* Purpose: frees up the memory taken by the buffer and its data

\* Author: William Collins (040652633)

\* History/Versions: 1.0

\* Called functions: free

\* Parameters: pB - a pointer to the buffer you want to destroy

\*/

void b\_destroy (Buffer \*pB)

{

/\* Make sure the buffer exists. If it doesn't, don't worry about it. Everyone

passes in a NULL pointer sometime in their life \*/

if (pB != NULL)

{

/\* Make sure it has data in it \*/

if (pB->ca\_head != NULL)

{

free(pB->ca\_head);

pB->ca\_head = NULL;

}

free(pB);

pB = NULL;

}

}

/\*

\* Purpose: Tests to see if the buffer is at capacity

\* Author: William Collins (040652633)

\* History/Versions: 1.0

\* Parameters: pB - a pointer to the buffer you want to test

\* Return value: returns -1 on error, 1 if the buffer is at capacity, and 0 otherwise

\*/

char ca\_isfull (Buffer \*pB)

{

/\*We say no to null\*/

if (pB == NULL)

{

return R\_FAIL\_1;

}

if (pB->addc\_offset == pB->capacity)

{

return 1;

}

return 0;

}

/\*

\* Purpose: gets the size of what is currently stored in the buffer

\* Author: William Collins (040652633)

\* History/Versions: 1.0

\* Parameters: pB - a pointer to the buffer you want the size of

\* Return value: returns -1 on error or the size of the buffer otherwise

\*/

int ca\_getsize (Buffer \*pB)

{

/\*Let's cut out this null pointer passing nonsense\*/

if (pB == NULL)

{

return R\_FAIL\_1;

}

return pB->addc\_offset;

}

/\*

\* Purpose: gets the capacity of the specified buffer

\* Author: William Collins (040652633)

\* History/Versions: 1.0

\* Parameters: pB - a pointer to the buffer in which you want the capacity of

\* Return value: returns -1 on error, otherwise it returns the capacity of the buffer

\*/

int ca\_getcapacity (Buffer \*pB)

{

/\*Are you null? We don't want you here if you are\*/

if (pB == NULL)

{

return R\_FAIL\_1;

}

return pB->capacity;

}

/\*

\* Purpose: Sets a particular mark in the buffer designating some point of interest

\* Author: William Collins (040652633)

\* History/Versions: 1.0

\* Parameters: pB - a pointer to the buffer in which to set the mark

mark - the offset from the beginning of the buffer data where the mark will point to

\* Return value: returns -1 on error, otherwise returns the mark

\*/

int ca\_setmark (Buffer \*pB, int mark)

{

/\*We hate null pointers\*/

if (pB == NULL)

{

return R\_FAIL\_1;

}

if (mark > pB->addc\_offset || mark <= 0)

{

return R\_FAIL\_1;

}

pB->mark\_offset = mark;

return mark;

}

/\*

\* Purpose: returns the mark of a buffer to the caller

\* Author: William Collins (040652633)

\* History/Versions: 1.0

\* Parameters: pB - a pointer to the buffer that holds the mark

\* Return value: -1 on error, otherwise returns the mark offset

\*/

int ca\_getmark (Buffer \*pB)

{

/\*Get these null pointers out of here\*/

if (pB == NULL)

{

return R\_FAIL\_1;

}

return pB->mark\_offset;

}

/\*

\* Purpose: returns the storage mode of the buffer to the caller

\* Author: William Collins (040652633)

\* History/Versions: 1.0

\* Parameters: pB - a pointer to the buffer you want the mofe of

\* Return value: -1 on error, otherwise returns the mode

\*/

int ca\_getmode (Buffer \*pB)

{

/\*We don't take kindly to null pointers around here\*/

if (pB == NULL)

{

return R\_FAIL\_1;

}

return pB->mode;

}

/\*

\* Purpose: shrinks the capacity of a buffer to the size of the data currently occupying it

\* Author: William Collins (040652633)

\* History/Versions: 1.0

\* Called functions: realloc

\* Parameters: pB - a pointer to the buffer that is to be packed

\* Return value: a pointer to the altered buffer, or NULL on error

\*/

Buffer \*ca\_pack( Buffer \*pB )

{

char \*temp;

/\*Null pointers aren't welcome here\*/

if (pB == NULL)

{

return NULL;

}

if (pB->ca\_head == NULL)

{

return NULL;

}

/\*Make sure that we don't have a weird offset\*/

if (pB->addc\_offset > pB->capacity || pB->addc\_offset < 0)

{

return NULL;

}

temp = realloc(pB->ca\_head, (sizeof(char) \* (pB->addc\_offset + 1 )));

if (temp == NULL)

{

return NULL;

}

if (temp != pB->ca\_head)

{

pB->r\_flag = 1;

}

pB->ca\_head = temp;

/\*Temp will go out of scope anyway, but we are practicing overly-defensive

programming, so why not protect against a dangling pointer here in case we

add more to the function eventually later on\*/

temp = NULL;

pB->capacity = (sizeof(char) \* (pB->addc\_offset + 1));

return pB;

}

/\*

\* Purpose: prints the data in the buffer to stdout

\* Author: William Collins (040652633)

\* History/Versions: 1.0

\* Called functions: printf

\* Parameters: pB - a pointer to the buffer that is to be printed

\* Return value: returns -1 on error, otherwise returns the number of characters printed

\*/

int ca\_print (Buffer \*pB)

{

int i;

if (pB == NULL)

{

return R\_FAIL\_1;

}

/\*Need to loop through the character array because there is no null terminator\*/

for (i = 0; i < pB->addc\_offset; i++)

{

printf("%c", pB->ca\_head[i]);

}

printf("\n");

return pB->addc\_offset;

}

/\*

\* Purpose: copy's a file of data into the buffer

\* Author: William Collins (040652633)

\* History/Versions: 1.0

\* Called functions: fgetc

\* Parameters: fi - a pointer to a file descriptor to a valid file containing data

pB - a pointer to the buffer that is to hold the data

\* Return value: returns -1 on error, LOAD\_FAIL if there is an error while copying, and otehrwise returns character count

\*/

int ca\_load (FILE \*fi,Buffer \*pB)

{

char temp; /\* Temporary character storage used for checking for end of file character \*/

if (pB == NULL || fi == NULL)

{

return R\_FAIL\_1;

}

if (pB->ca\_head == NULL)

{

return R\_FAIL\_1;

}

/\* copy over the contents of the file \*/

while (!feof(fi))

{

if ((temp = (char) fgetc(fi)) != EOF )

{

if ( (pB = ca\_addc(pB, temp) ) == NULL)

{

return LOAD\_FAIL;

}

}

}

return pB->addc\_offset;

}

**ass1fi.out**

Reading file ass1.pls ....Please wait

The input file ass1.pls is not completely loaded.

Input file size: 327

Printing buffer parameters:

The capacity of the buffer is: 200

The current size of the buffer is: 200

The operational mode of the buffer is: 0

The increment factor of the buffer is: 0

The current mark of the buffer is: 200

Printing buffer contents:

Compilers are fundamental to modern computing.

They act as translators, transforming human-oriented

language into computer-oriented machine-language.

A compiler allows virtually all compu

Printing buffer parameters:

The capacity of the buffer is: 201

The current size of the buffer is: 201

The operational mode of the buffer is: 0

The increment factor of the buffer is: 0

The current mark of the buffer is: 200

Printing buffer contents:

Compilers are fundamental to modern computing.

They act as translators, transforming human-oriented

language into computer-oriented machine-language.

A compiler allows virtually all compuÿ

**ass1ai.out**

Reading file ass1.pls ....Please wait

Printing buffer parameters:

The capacity of the buffer is: 320

The current size of the buffer is: 319

The operational mode of the buffer is: 1

The increment factor of the buffer is: 15

The current mark of the buffer is: 319

Printing buffer contents:

Compilers are fundamental to modern computing.

They act as translators, transforming human-oriented

language into computer-oriented machine-language.

A compiler allows virtually all computer users to

ignore the machine-dependent details of machine language.

Isn't that nice? =:)

CST8152

Summer, 2011

Printing buffer parameters:

The capacity of the buffer is: 320

The current size of the buffer is: 320

The operational mode of the buffer is: 1

The increment factor of the buffer is: 15

The current mark of the buffer is: 319

Printing buffer contents:

Compilers are fundamental to modern computing.

They act as translators, transforming human-oriented

language into computer-oriented machine-language.

A compiler allows virtually all computer users to

ignore the machine-dependent details of machine language.

Isn't that nice? =:)

CST8152

Summer, 2011ÿ

**ass1mi.out**

Reading file ass1.pls ....Please wait

Printing buffer parameters:

The capacity of the buffer is: 5085

The current size of the buffer is: 319

The operational mode of the buffer is: -1

The increment factor of the buffer is: 15

The current mark of the buffer is: 319

Printing buffer contents:

Compilers are fundamental to modern computing.

They act as translators, transforming human-oriented

language into computer-oriented machine-language.

A compiler allows virtually all computer users to

ignore the machine-dependent details of machine language.

Isn't that nice? =:)

CST8152

Summer, 2011

Printing buffer parameters:

The capacity of the buffer is: 320

The current size of the buffer is: 320

The operational mode of the buffer is: -1

The increment factor of the buffer is: 15

The current mark of the buffer is: 319

Printing buffer contents:

Compilers are fundamental to modern computing.

They act as translators, transforming human-oriented

language into computer-oriented machine-language.

A compiler allows virtually all computer users to

ignore the machine-dependent details of machine language.

Isn't that nice? =:)

CST8152

Summer, 2011ÿ

**ass1e.out**

Reading file ass1e.pls ....Please wait

Printing buffer parameters:

The capacity of the buffer is: 200

The current size of the buffer is: 0

The operational mode of the buffer is: 0

The increment factor of the buffer is: 0

The current mark of the buffer is: 0

Printing buffer contents:

Printing buffer parameters:

The capacity of the buffer is: 1

The current size of the buffer is: 1

The operational mode of the buffer is: 0

The increment factor of the buffer is: 0

The current mark of the buffer is: 0

Printing buffer contents:

Ÿ

**Fixed.out**

Reading file buffer.c ....Please wait

Printing buffer parameters:

The capacity of the buffer is: 10220

The current size of the buffer is: 10214

The operational mode of the buffer is: 1

The increment factor of the buffer is: 15

The current mark of the buffer is: 10214

Printing buffer contents:

/\* File Name: buffer.c

\* Version: 1.0

\* Author: William Collins (040652633)

\* Course: CST8152 - Compilers

\* Assignment: 1

\* Date: May 18, 2011

\* Professor: Svillen Ranev

\* Purpose: Provide a utility to create and manage a buffer

\* Function list: b\_create, ca\_addc, b\_reset, b\_destroy, ca\_isfull,

ca\_getsize, ca\_getcapacity, ca\_setmark, ca\_getmark,

ca\_getmode, ca\_pack, ca\_print, ca\_load

\*/

#include "buffer.h"

…etc

Printing buffer parameters:

The capacity of the buffer is: 10215

The current size of the buffer is: 10215

The operational mode of the buffer is: 1

The increment factor of the buffer is: 15

The current mark of the buffer is: 10214

..etc