Compilers Assignment #1:

The Buffer

|  |  |
| --- | --- |
| **Student Name** | Seongyeop Jeong (040 885 882) |
| **Course** | CST8152 – Compilers |
| **Lab Section** | 012 |
| **Assignment** | 1 |
| **Professor** | Sv. Ranev |
| **Due date** | October 3, 2018 |
| **Date** | October 3, 2018 |

**Contents**

* Assignment #1 Test Plan
* Printed source code
  + buffer.h
  + buffer.c
* Printed test file
  + ass1ai.out
  + ass1e.out
  + ass1fi.out
  + ass1mi.out

**Assignment Test Plan**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Course | | CST8152 – Compilers | | |
| Lab Section | | 012 | | |
| Assignment | | 1 | | |
| Professor | | Sv. Ranev | | |
| Date | | October 3, 2018 | | |
| **#** | **Test file** | **Mode** | **Value:**  **init\_capacity** | **Value:**  **Inc\_factor** | **Matched result** | **Command** |
| 01 | ass1ai.out | Additive | 200 | 15 | Yes | buffer.exe ass1.pls additive > ass1ai.out |
| 02 | ass1mi.out | Multiplicative | 200 | 15 | Yes | buffer.exe ass1.pls multiplicative > ass1mi.out |
| 03 | ass1fi.out | Fixed | 200 | 15 | Yes | buffer.exe ass1.pls fixed > ass1fi.out |
| 04 | ass1e.out | Fixed | 200 | 15 | Yes | buffer.exe ass1.pls fixed > ass1e.out |
| 05 | bigfai.out | Additive | 200 | 15 | Yes | buffer.exe bigf.pls additive > bigfai.out |
| 06 | bigfai128.out | Additive | 200 | 128 | Yes | buffer.exe bigf.pls additive > bigfai128.out |
| 07 | bigfamax.out | Additive | SHRT\_MAX-1 | 2 | Yes | buffer.exe bigf.pls additive > bigfamax.out |
| 08 | bigffi0.out | Fixed | 200 | 0 / any | Yes | buffer.exe bigf.pls fixed > bigffi0.out |
| 09 | bigffi128.out | Fixed | 200 | 128 / any | Yes | buffer.exe bigf.pls fixed > bigffi128.out |
| 10 | bigfmi.out | Multiplicative | 200 | 15 | Yes | buffer.exe bigf.pls multiplicative > bigfmi.out |
| 11 | bigfmi9.out | Multiplicative | SHRT\_MAX-10 | 9 | Yes | buffer.exe bigf.pls multiplicative > bigfmi9.out |
| 12 | bigfmmax.out | Multiplicative | SHRT\_MAX-1 | 15 | Yes | buffer.exe bigf.pls multiplicative > bigfmmax.out |
| 13 | bigfai0.out | Additive | 200 | 0 | Yes | buffer.exe bigf.pls additive > bigfai0.out |
| 14 | bigfmi0.out | Multiplicative | 200 | 0 | Yes | buffer.exe bigf.pls multiplicative > bigfmi0.out |
| 15 | bigfac0.out | Additive | 0 | 15 | Yes | buffer.exe bigf.pls additive > bigfac0.out |
| 16 | bigfmaxc.out | Any | SHRT\_MAX | any | Yes | buffer.exe bigf.pls additive > bigfmaxc.out  buffer.exe bigf.pls multiplicative > bigfmaxc.out  buffer.exe bigf.pls fixed > bigfmaxc.out |
| 17 | bigfmil128.out | Multiplicative | 200 | > 100 | Yes | buffer.exe bigf.pls multiplicative > bigfmil128.out |
| 18 | bigffc0.out | Fixed | 0 | any | Yes | buffer.exe bigf.pls fixed > bigffc0.out |
| 19 | bigfc-1.out | Any | -1 | any | Yes | buffer.exe bigf.pls additive > bigfc-1.out  buffer.exe bigf.pls multiplicative > bigfc-1.out  buffer.exe bigf.pls fixed > bigfc-1.out |

**Comments**

All the test cases do work successfully. Every case has been tested with debug by step by step when it caused problem. When it has not been solved problems, I thought about buffer shape and flags deeply so that it has been worked successfully.

Although every case is tested in detail, I am not sure if the case number 15 is right. Because the tested file which is provided from professor handle only additive case so that I could not test multiplicative case.

**Buffer header file**

/\*

\* File Name : buffer.h

\* Compiler : MS Visual Studio 2015

\* Author : Seongyeop Jeong, ID# 040885882

\* Course : CST8152 - Compiler, Lab Section: 012

\* Assignment : 1

\* Date : 2 October 2018

\* Professor : Svillen Ranev

\* Purpose : This file is to implement your source code to parse the bitwise.

\* function list : b\_allocate, b\_addc, b\_clear, b\_free, b\_isfull, b\_limit, b\_capacity, b\_mark,

\* b\_mode, b\_incfactor, b\_load, b\_isempty, b\_getc, b\_eob, b\_print, b\_compact,

\* b\_rflag, b\_retract, b\_reset, b\_getcoffset, b\_rewind, b\_location

\*/

#ifndef BUFFER\_H\_

#define BUFFER\_H\_

/\*#pragma warning(1:4001) \*//\*to enforce C89 type comments - to make //comments an warning \*/

/\*#pragma warning(error:4001)\*//\* to enforce C89 comments - to make // comments an error \*/

/\* standard header files \*/

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

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

/\* definition of the B\_FULL macro \*/

#define B\_FULL

/\* constant definitions \*/

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

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

#define LOAD\_FAIL -2 /\* load fail return value \*/

/\* You should add your own constant definitions here \*/

#define TRUE 1

#define FALSE 0

#define O\_MODE\_ZERO 0

#define O\_MODE\_ONE 1

#define O\_MODE\_MINUS\_ONE -1

#define INC\_FACTOR\_ZERO 0

#define INC\_FACTOR\_MIN 1

#define INC\_FACTOR\_HUNDRED 100

#define INC\_FACTOR\_MAX 255

#define OFFSET\_RESET 0

/\* Enter your bit-masks constant definitions here \*/

#define ERR\_INC\_FACTOR 0x100

#define DEFAULT\_FALGS 0xFFFC /\* default flags value: 1111 1111 1111 1100 \*/

#define SET\_EOB 0x0001 /\* set eob mask: 0001 \*/

#define RESET\_EOB 0xFFFE /\* reset eob mask: 1111 1111 1111 1110 \*/

#define CHECK\_EOB 0x0001 /\* check eob mask: same with set \*/

#define SET\_R\_FLAG 0x0002 /\* set r\_flag mask: 0010 \*/

#define RESET\_R\_FLAG 0xFFFD /\* reset r\_flag mask: 1111 1111 1111 1101 \*/

#define CHECK\_R\_FLAG 0x0002 /\*check r\_flag mask: 0010 \*/

/\* user data type declarations \*/

typedef struct BufferDescriptor {

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

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

short addc\_offset; /\* the offset (in chars) to the add-character location \*/

short getc\_offset; /\* the offset (in chars) to the get-character location \*/

short markc\_offset; /\* the offset (in chars) to the mark location \*/

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

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

unsigned short flags; /\* contains character array reallocation flag and end-of-buffer flag \*/

} Buffer, \*pBuffer;

/\* function declarations \*/

/\*

Place your function declarations here.

Do not include the function header comments here.

Place them in the buffer.c file

\*/

Buffer \* b\_allocate(short init\_capacity, char inc\_factor, char o\_mode);

pBuffer b\_addc(pBuffer const pBD, char symbol);

int b\_clear(Buffer \*const pBD);

void b\_free(Buffer \* const pBD);

int b\_isfull(Buffer \* const pBD);

short b\_limit(Buffer \*const pBD);

short b\_capacity(Buffer \*const pBD);

short b\_mark(pBuffer const pBD, short mark);

int b\_mode(Buffer \* const pBD);

size\_t b\_incfactor(Buffer \* const pBD);

int b\_load(FILE \* const fi, Buffer \* const pBD);

int b\_isempty(Buffer \* const pBD);

char b\_getc(Buffer \* const pBD);

int b\_eob(Buffer \* const pBD);

int b\_print(Buffer \* const pBD);

Buffer \* b\_compact(Buffer \* const pBD, char symbol);

char b\_rflag(Buffer\* const pBD);

short b\_retract(Buffer \* const pBD);

short b\_reset(Buffer \* const pBD);

short b\_getcoffset(Buffer \* const pBD);

int b\_rewind(Buffer \* const pBD);

char \* b\_location(Buffer \* const pBD, short loc\_offset);

#endif

/\* macro for b\_isfull \*/

#ifdef B\_FULL

#define b\_isfull(pBD) ((pBD==NULL) ? \

RT\_FAIL\_1 : \

((pBD->addc\_offset == pBD->capacity) ? \

TRUE : FALSE))

#elif !defined B\_FULL

#undef B\_FULL

#endif

**Buffer source code**

/\*

\* File Name : buffer.c

\* Compiler : MS Visual Studio 2015

\* Author : Seongyeop Jeong, ID# 040885882

\* Course : CST8152 - Compiler, Lab Section: 012

\* Assignment : 1

\* Date : 2 October 2018

\* Professor : Svillen Ranev

\* Purpose : This file is to implement source code to parse the bitwise on the buffer

\* function list : b\_allocate, b\_addc, b\_clear, b\_free, b\_isfull, b\_limit, b\_capacity, b\_mark,

\* b\_mode, b\_incfactor, b\_load, b\_isempty, b\_getc, b\_eob, b\_print, b\_compact,

\* b\_rflag, b\_retract, b\_reset, b\_getcoffset, b\_rewind, b\_location

\*/

#include <stdlib.h>

#include "buffer.h"

/\*

\* Purpose: This method is to allocate new memory with malloc on the heap.

it will be different with increment values depending on what mode do user want to see.

(mode: fixed, additive, multiplicative)

\* Author: Seongyeop Jeong

\* Versions: 1.0

\* Called Functions: calloc(), malloc(), free()

\* Parameters: short init\_capacity, char inc\_factor, char o\_mode

\* Return value: pBD, NULL

\* Algorithm: 1. allocate memory for one Buffer;

\* 2. allocates memory for one dynamic character buffer;

\* 3. sets the buffer operational mode indicator mode

\*/

Buffer \* b\_allocate(short init\_capacity, char inc\_factor, char o\_mode)

{

/\* allocate memory for one Buffer descriptor \*/

pBuffer pB = (Buffer \*)calloc(1, sizeof(Buffer));

/\* If the pBuffer declare, it might be caused NULL even though pB is dereferenced.

so that it has to be returned NULL to avoid run-time error when pBuffer has null value \*/

if (pB == NULL)

{

return NULL;

}

/\* to avoid mistake, assigned unsigned char inc\_factor to the variable \*/

unsigned char unsignedIncFactor = (unsigned char)inc\_factor;

/\* reset the flags to the dafulat flags \*/

pB->flags |= DEFAULT\_FALGS;

/\* allocates memory for one dynamic character buffer \*/

pB->cb\_head = (char \*)malloc(sizeof(char) \* init\_capacity);

/\* if init\_capacity is smaller then maximum value, and it is greater than 0,

application has to launch. if it is not, abort and release the memory by using free.

then return null \*/

if (init\_capacity < SHRT\_MAX && init\_capacity >= 0)

{

/\* sets the buffer operational mode indicator mode 'f' \*/

if (o\_mode == 'f' || (unsignedIncFactor == INC\_FACTOR\_ZERO) || o\_mode == 'f' && (unsignedIncFactor != INC\_FACTOR\_ZERO))

{

/\* if init\_capacity is 0, return NULL to avoid run-time error \*/

if (init\_capacity == 0)

return NULL;

/\* fixed mode is 0 \*/

pB->mode = O\_MODE\_ZERO;

/\* increment factor is 0 in fixed mode only \*/

pB->inc\_factor = INC\_FACTOR\_ZERO;

}

/\* in operational mode 'a', unsigned inc\_factor is greater than 1, and smaller than 255 \*/

else if (o\_mode == 'a' && (unsignedIncFactor >= INC\_FACTOR\_MIN) && (unsignedIncFactor <= INC\_FACTOR\_MAX))

{

/\* 'a' mode as in number 1 \*/

pB->mode = O\_MODE\_ONE;

/\* assign the unsigned increment factor to inc\_factor of the buffer \*/

pB->inc\_factor = unsignedIncFactor;

}

/\* in operational mode 'm', unsigned inc\_factor is greater than 1, and smaller then 100 \*/

else if (o\_mode == 'm' && (unsignedIncFactor >= INC\_FACTOR\_MIN) && (unsignedIncFactor <= INC\_FACTOR\_HUNDRED))

{

/\* 'a' mode as in number -1 \*/

pB->mode = O\_MODE\_MINUS\_ONE;

/\* unsigned inc\_factor has to be assigned in inc\_factor of the buffer \*/

pB->inc\_factor = unsignedIncFactor;

}

/\* abort the buffer by using free function and return NULL. \*/

else {

/\* free a pointer of the memory of the buffer firstly \*/

free(pB->cb\_head);

/\* then free the struct of the buffer \*/

free(pB);

return NULL;

}

}

/\* otherwise, abort and release memory \*/

else

{

free(pB->cb\_head);

free(pB);

return NULL;

}

/\* if allocation memory fails, abort and release it \*/

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

{

free(pB->cb\_head);

free(pB);

return NULL;

}

/\* copies the given init\_capacity value into the Buffer structure capacity variable. \*/

pB->capacity = init\_capacity;

return pB;

}

/\*

\* Purpose: The function resets r\_flag to 0 and decide how many space it apply depending on the mode

then tries to add the character symbol to the character array which come through buffer pointed by pBD

if the reallocation is successful, this will be set r\_flag to 1 with bitwise operation

\* Author: Seongyeop Jeong

\* Versions: 1.0

\* Called Functions: realloc()

\* Parameters: pBuffer const pBD, char symbol

\* Return value: pBuffer, NULL

\*

\* Algorithm: 1. if the buffer is operational and not full, add the character to buffer

\* 2. if the buffer is full, resize the buffer depending on the mode (a, m, f)

\* 3. if the operational mode, function returns different value

\* operational mode indicator mode

\*/

pBuffer b\_addc(pBuffer const pBD, char symbol)

{

/\* to assign the capacity because it depends on the mode \*/

unsigned short newCapacity = 0;

/\* to get available space \*/

short available\_space = 0;

/\* this increment variable is for mode multiplicative \*/

unsigned short newInc = 0;

/\* reallocation memory to this variable temporary.

it will be assigned to pBD->cb\_head after some conditions are satisfied\*/

char\* temp;

/\* to avoid run-time error \*/

if (pBD == NULL)

return NULL;

/\* flags field reset the allocation \*/

pBD->flags &= RESET\_R\_FLAG;

/\* this 'if statement' is to expand the capacity.

if there is no any space, do progress depending on mode type \*/

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

{

/\* if mode is 0, it has to return null because fixed buffer doesn't expand the capacity \*/

if (pBD->mode == O\_MODE\_ZERO)

return NULL;

/\* mode is 1 (additive) \*/

else if (pBD->mode == O\_MODE\_ONE)

{

/\* if the result is positive and does not exceed minus 1, this function proceeds\*/

if (pBD->capacity < 0 || pBD->capacity >= SHRT\_MAX - 1)

return NULL;

/\* assign the capacity adding inc\_factor to newCapcity to expand buffer.

new capacity has to be unsgiend short so that it has to be casted to unsigned in calculation \*/

newCapacity = (unsigned short)pBD->capacity + (unsigned char)pBD->inc\_factor;

/\* if the newCapacity is greater than maximum number -1, it has to be stop \*/

if (newCapacity >= SHRT\_MAX - 1)

{

/\* so that maximum number-1 is assigned to new capacity \*/

newCapacity = SHRT\_MAX - 1;

/\* then return NULL, to display and let main program know it is full \*/

return NULL;

}

} /\* end of the if function: mode 1 \*/

/\* if the mode -1, incearse the current capacity of the buffer to a new capacity\*/

else if (pBD->mode == O\_MODE\_MINUS\_ONE)

{

/\* if current capacity cannot be incremented anymore

because it has reached the maximum capacity of the buffer,

the function returns NULL \*/

if (pBD->capacity == SHRT\_MAX - 1)

return NULL;

/\* to get available space, current capacity has to be deducted from maximum number - 1 \*/

available\_space = SHRT\_MAX - 1 - pBD->capacity;

/\* after calculating the increments, it has to be casted to unsigned to make them as positive number \*/

newInc = (unsigned short)(available\_space \* (unsigned char)pBD->inc\_factor / 100.0f);

/\* validation for new capacity: total value does not have to be greater than maximum number -1

and also new increment must not 0 \*/

if (pBD->capacity + newInc < SHRT\_MAX - 1 && newInc != 0)

/\* if so, assign to new capacity to expand the buffer \*/

newCapacity = (unsigned short)pBD->capacity + newInc;

/\* it is to avoid over flow from SHRT\_MAX -1 \*/

else if (pBD->capacity < SHRT\_MAX - 1)

newCapacity = SHRT\_MAX - 1;

} /\* end of if function : mode -1 \*/

else

return NULL;

/\* If the capacity increment in mode 1 or -1 is successful, reallocate memory

expand the character buffer calling realloc() with the new capacity \*/

temp = (char \*)realloc(pBD->cb\_head, sizeof(char)\*newCapacity);

/\* If reallocation fails, returns NULL \*/

if (temp == NULL)

return NULL;

else /\* if the reallocation is successful, \*/

{

/\* whenever program reallocation, the memory location is changed so that it has to be set flags \*/

if (temp != pBD->cb\_head)

pBD->flags |= SET\_R\_FLAG; /\* sets r\_flag bit to 1 \*/

}

pBD->cb\_head = temp;

pBD->capacity = newCapacity;

}

/\* if there is enough space, adds the char symbol to the bufer content\*/

if (pBD->addc\_offset < pBD->capacity)

pBD->cb\_head[pBD->addc\_offset++] = symbol;

/\* if not, return null to pass to main program \*/

else

return NULL;

return pBD;

}

/\*

\* Purpose: This function is to reset the offsets to clean the memory locations currently buffer struct

such as addc, getc, and markc.

\* Author: Seongyeop Jeong

\* Versions: 1.0

\* Called Functions: none

\* Parameters: Buffer\* const pBD

\* Return value: -1, 1

\* Algorithm: 1. when the code is reached to the b\_clear method, this method is to reset the offsets

\* 2. if there is no run-time error, it returns TRUE which is 1 (not 0)

\*/

int b\_clear(Buffer \* const pBD) {

/\* If run-time error ocurrs, returns -1 \*/

if (pBD == NULL)

return RT\_FAIL\_1;

/\* reinitialize the data members of buffer to 0 \*/

pBD->addc\_offset &= OFFSET\_RESET;

pBD->getc\_offset &= OFFSET\_RESET;

pBD->markc\_offset &= OFFSET\_RESET;

/\* except for 0 (false), every numbers are TRUE, here's true is defined to 1 \*/

return TRUE;

}

/\*

\* Purpose: This function is to abort and release memory of the Buffer struct when this application is done.

\* Author: Seongyeop Jeong

\* Versions: 1.0

\* Called Functions: free()

\* Parameters: Buffer\* const pBD

\* Return value:

\* Algorithm: 1. if pointer of the buffer is not null, memory locations have to be released.

\*/

void b\_free(Buffer \*const pBD)

{

if (pBD != NULL)

{

/\* free the pointer of the buffer firstly, then free memory of the buffer.\*/

free(pBD->cb\_head);

free(pBD);

}

}

/\*

\* Purpose: This function is to check if the character on the buffer is full. then it returns false, true or -1

\* Author: Seongyeop Jeong

\* Versions: 1.0

\* Called Functions: none

\* Parameters: Buffer\* const pBD

\* Return value: -1, 1, 0

\* Algorithm: 1. check the run-time error. if there is, return -1.

\* 2. check if addc\_offset and capacity are equal, return true(every number except for 0).

\* 3. if not, return false(0).

\*/

#ifndef B\_FULL

int b\_isfull(Buffer \*const pBD)

{

if (pBD == NULL)

return RT\_FAIL\_1;

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

return TRUE;

else

return FALSE;

}

#endif

/\*

\* Purpose: This function is to returns current limit size of buffer to display it.

\* Author: Seongyeop Jeong

\* Versions: 1.0

\* Called Functions: none

\* Parameters: Buffer\* const pBD

\* Return value: addc\_offset

\* Algorithm: 1. If there is no run-time error, return addc\_offset from buffer struct.

\*/

short b\_limit(Buffer \*const pBD)

{

if (pBD == NULL)

return RT\_FAIL\_1;

return pBD->addc\_offset;

}

/\*

\* Purpose: This function is to return current capacity on the buffer to display it.

\* Author: Seongyeop Jeong

\* Versions: 1.0

\* Called Functions: none

\* Parameters: Buffer\* const pBD

\* Return value: -1, capacity

\* Algorithm: 1. if there is no run-time error, it returns capacity.

\*/

short b\_capacity(Buffer \*const pBD)

{

if (pBD == NULL)

return RT\_FAIL\_1;

return pBD->capacity;

}

/\*

\* Purpose: This function is to return the current limit size of the buffer

\* Author: Seongyeop Jeong

\* Versions: 1.0

\* Called Functions: none

\* Parameters: Buffer const pBD, short mark

\* Return value: -1, markc\_offset

\* Algorithm: 1. if there is no run-time error, do process.

\* 2. if mark value is smaller than 0 then it is greater than addc\_offset, it returns -1

\* 3. then mark assigns to markc\_offset of buffer

\*/

short b\_mark(pBuffer const pBD, short mark)

{

if (pBD == NULL)

return RT\_FAIL\_1;

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

return RT\_FAIL\_1;

pBD->markc\_offset = mark;

return pBD->markc\_offset;

}

/\*

\* Purpose: This function is to set up what mode this application run with.

\* Author: Seongyeop Jeong

\* Versions: 1.0

\* Called Functions: none

\* Parameters: Buffer\* const pBD

\* Return value: -1, 1, 0

\* Algorithm: 1. if there is no run-time error, it returns mode value.

\*/

int b\_mode(Buffer \* const pBD)

{

/\* run-time error, this function notify the calling function about the failure \*/

if (pBD == NULL)

return LOAD\_FAIL;

return pBD->mode;

}

/\*

\* Purpose: This function is to return the increment factor which is unsigned value.(non-negative value)

\* Author: Seongyeop Jeong

\* Versions: 1.0

\* Called Functions: none

\* Parameters: Buffer\* const pBD

\* Return value: 0x100, inc\_factor

\* Algorithm: 1. if there is no run-time error, do process.

\* 2. returns inc factor which is unsigned char.

\*/

size\_t b\_incfactor(Buffer \* const pBD)

{

if (pBD == NULL)

return ERR\_INC\_FACTOR;

return (unsigned char)pBD->inc\_factor;

}

/\*

\* Purpose: This function is to return the number of characters on the buffer by getting a character .

\* Author: Seongyeop Jeong

\* Versions: 1.0

\* Called Functions: fgetc(), feof(), ungetc(), b\_addc(), printf()

\* Parameters: FILE\* const fi, Buffer\* const pBD

\* Return value: -1, -2, charNum

\* Algorithm: 1. check if there is run-time error

\* 2. each character will be added to the buffer through b\_addc method until the end of the file

\* 3. if the b\_addc method's return is not null,

\* count up the number of characters how many character does it work with the method

\* 4. if the b\_addc method's return is null,

\* print it what the last character was.

\*/

int b\_load(FILE \* const fi, Buffer \* const pBD)

{

/\* actually this variable is not buffer. it is to get a character from fgetc \*/

char buffer;

int charNum = 0;

if (fi == NULL || pBD == NULL)

return RT\_FAIL\_1;

while (1) /\* repeat until the standard macro feof(fi)detects EOF \*/

{

/\* fgetc function is to get a character \*/

buffer = (char)fgetc(fi);

/\* feof function: tests the end-of-file indicator,

it will be returned non-zero when end-of-file indicator

associated with the stream is set \*/

if (feof(fi))

break;

/\* check the buffer by passing the buffer and a character \*/

if (b\_addc(pBD, buffer) == NULL)

{

/\* the last character will be pushed on the buffer stream \*/

ungetc(buffer, fi);

/\* print out the character which is last one on the stream \*/

printf("The last character read from the file is: %c %d\n", buffer, buffer);

return LOAD\_FAIL;

}

/\* if b\_addc != null, count character on the buffer by 1 \*/

charNum++;

}

return (int)charNum;

}

/\*

\* Purpose: This function is to check if the buffer's addc\_offset is 0 or not.

\* Author: Seongyeop Jeong

\* Versions: 1.0

\* Called Functions: none

\* Parameters: Buffer\* const pBD

\* Return value: -1, 1, 0

\* Algorithm: 1. check if the run-time erorr is happened

\* 2. check if the addc\_offset is 0, then return 1.

\* 3. check if any else situation will be return 0.

\*/

int b\_isempty(Buffer \* const pBD)

{

if (pBD == NULL)

return RT\_FAIL\_1;

/\* if the addc\_offset is 0, the function returns 1; otherwise it returns 0 \*/

if (pBD->addc\_offset == 0)

return TRUE;

else

return FALSE;

}

/\*

\* Purpose: This function is to get character of the buffer.

When the addc\_offset is same with the getc\_offset,

set the eob to 1 bit to make sure that it has been reached current limit on the buffer

\* Author: Seongyeop Jeong

\* Versions: 1.0

\* Called Functions: none

\* Parameters: Buffer\* const pBD

\* Return value: -2, 0, pointer of the buffer

\* Algorithm: 1. in b\_print method, b\_getc will be checked

\* until it will catch them if getc and addc are equal.

\* 2. when the getc and addc are equal,

\* set the EOB to 1 bit, and returns 0.

\* 3. if not, reset the eob to 0 bit.

\* Then next pointer of getc\_offset of the buffer will be returned

\*/

char b\_getc(Buffer \* const pBD)

{

if (pBD == NULL)

return RT\_FAIL\_2;

/\* if the getc and addc are equal, the current buffer has been reached on the buffer limit \*/

if (pBD->getc\_offset == pBD->addc\_offset)

{

/\* set to 1 bit. End of the buffer is 1 because the buffer is full \*/

pBD->flags |= SET\_EOB;

return 0;

}

/\* reset the eob to 0 bit so that it has not been reached current limit on the buffer

then do progress to the next position of the buffer \*/

else

pBD->flags &= RESET\_EOB;

return pBD->cb\_head[pBD->getc\_offset++];

}

/\*

\* Purpose: This function is to return value to check in b\_print method if it is end of the buffer

\* Author: Seongyeop Jeong

\* Versions: 1.0

\* Called Functions: none

\* Parameters: Buffer\* const pBD

\* Return value: -1, return flags which is checked 0x0001

\* Algorithm: 1. make a new variable to check if it is end of buffer.

\* 2. then returns the variable to the b\_print method

\*/

int b\_eob(Buffer \* const pBD)

{

/\* to avoid run-time error \*/

if (pBD == NULL)

return RT\_FAIL\_1;

/\* to check if it is end of the buffer for method of b\_print \*/

short checkEob;

checkEob = pBD->flags & CHECK\_EOB;

return checkEob;

}

/\*

\* Purpose: This function is to print out the characters on the buffer to the standard output.

\* Author: Seongyeop Jeong

\* Versions: 1.0

\* Called Functions: b\_getc(), b\_eob(), printf()

\* Parameters: Buffer\* const pBD

\* Return value: -1, 0, getc\_offset

\* Algorithm:

\* 1. if addc\_offset of the buffer struct is 0, returns 0.

\* 2. if not, it gets a character by using b\_getc method

\* then prints out until it reachs to end of the buffer.

\* 3. it returns getc\_offset of buffer struct which is from b\_getc method.

\*/

int b\_print(Buffer \* const pBD)

{

/\* to get a character by using the b\_getc method \*/

char buffer;

/\* avoid the run-time error \*/

if (pBD == NULL)

return RT\_FAIL\_1;

/\* if the addc\_offset of the buffer 0, print out empty buffer, then returns 0 \*/

if (pBD->addc\_offset == FALSE)

{

printf("Empty buffer!");

printf("\n");

return 0;

}

while (1)

{

/\* get a character \*/

buffer = b\_getc(pBD);

/\* if it is end of the buffer, break. \*/

if (b\_eob(pBD))

break;

printf("%c", buffer);

}

printf("\n");

return pBD->getc\_offset;

}

/\*

\* Purpose: This function is to compact the space for one more character when it is full.

\* Author: Seongyeop Jeong

\* Versions: 1.0

\* Called Functions: realloc(),

\* Parameters: Buffer\* const pBD, char symbol

\* Return value: -1, NULL, Buffer

\* Algorithm: 1. realloc memory location into next position on the buffer

\* 2. if the temp is not matched to head of the buffer, set the r\_flag

\*/

Buffer \* b\_compact(Buffer \* const pBD, char symbol)

{

/\* declare the pointer temp to reallocation memory temporary \*/

char\* temp;

/\* avoid the run-time error \*/

if (pBD == NULL)

return NULL;

/\* uses realloc to adjust the new capacity \*/

temp = (char\*)realloc(pBD->cb\_head, pBD->addc\_offset + 1);

/\* avoid the run-time error if the temp fail the reallocation \*/

if (temp == NULL)

return NULL;

/\* if there is enough space in buffer, add 1 to offset \*/

if (pBD->capacity <= SHRT\_MAX - 1)

pBD->capacity = pBD->addc\_offset + 1;

/\* to the pointer of the buffer, temp has to be assigned \*/

pBD->cb\_head = temp;

/\* the function adds the symbol to the end of the char buffer \*/

pBD->cb\_head[pBD->addc\_offset++] = symbol;

/\* after reallocation, it is possible to be changed the memory location

so that it must set the r\_flag bit 1 \*/

if (temp != pBD->cb\_head)

pBD->flags |= SET\_R\_FLAG;

return pBD;

}

/\*

\* Purpose: This function is to set up the r\_flag with bitwise operator.

\* Author: Seongyeop Jeong

\* Versions: 1.0

\* Called Functions: none

\* Parameters: Buffer\* const pBD

\* Return value: -1, flags which is set r\_flag to 1

\* Algorithm: 1. return reseted r\_flag to the flag of buffer

\*/

char b\_rflag(Buffer\* const pBD)

{

if (pBD == NULL)

return RT\_FAIL\_1;

return pBD->flags &= SET\_R\_FLAG;

}

/\*

\* Purpose: This function is to decrements getc\_offset by 1, however it has been used in this assignment

\* Author: Seongyeop Jeong

\* Versions: 1.0

\* Called Functions: none

\* Parameters: Buffer\* const pBD

\* Return value: decremented getc\_offset of the buffer

\* Algorithm: 1. return decremented getc\_offset by 1

\*/

short b\_retract(Buffer \* const pBD)

{

if (pBD == NULL)

return RT\_FAIL\_1;

else

return pBD->getc\_offset;

return pBD->getc\_offset--;

}

/\*

\* Purpose: This function is to reset the getc\_offset to the current markc\_offset.

\* Author: Seongyeop Jeong

\* Versions: 1.0

\* Called Functions: none

\* Parameters: Buffer\* const pBD

\* Return value: -1, getc\_offset from Buffer struct

\* Algorithm: 1. reset the getc\_offset to markc\_offset

\*/

short b\_reset(Buffer \* const pBD)

{

pBD->getc\_offset = pBD->markc\_offset;

/\* to avoid run-time error \*/

if (pBD == NULL || pBD->markc\_offset < OFFSET\_RESET)

return RT\_FAIL\_1;

else

return pBD->getc\_offset;

}

/\*

\* Purpose: This function is to get the current getc\_offsets

\* Author: Seongyeop Jeong

\* Versions: 1.0

\* Called Functions: none

\* Parameters: Buffer\* const pBD

\* Return value: -1, getc\_offset from buffer struct

\* Algorithm: 1. return the current getc\_offset of the buffer

\*/

short b\_getcoffset(Buffer \* const pBD)

{

if (pBD == NULL)

return RT\_FAIL\_1;

return pBD->getc\_offset;

}

/\*

\* Purpose: This function is to set the offsets to 0 such as getc and markc.

\* Author: Seongyeop Jeong

\* Versions: 1.0

\* Called Functions: none

\* Parameters: Buffer\* const pBD

\* Return value: -1, 0

\* Algorithm: 1. assign 0 to getc and markc offsets.

\*/

int b\_rewind(Buffer \* const pBD)

{

pBD->getc\_offset = 0;

pBD->markc\_offset = 0;

if (pBD == NULL)

return RT\_FAIL\_1;

else

return 0;

}

/\*

\* Purpose: This function is to return a pointer of the buffer by adding location of the character buffer

\* to get the begining of the character array. However, it has not been used in this assignment.

\* Author: Seongyeop Jeong

\* Versions: 1.0

\* Called Functions: none

\* Parameters: Buffer\* const pBD

\* Return value: NULL, pointer of buffer struct

\* Algorithm: 1. return a pointer to a location of the chacracter buffer

\*/

char \* b\_location(Buffer \* const pBD, short loc\_offset)

{

if (pBD == NULL)

return NULL;

else if (pBD->addc\_offset <= loc\_offset || loc\_offset >= 0)

return NULL;

return pBD->cb\_head + loc\_offset;

}

**test file: 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

The value of the flags field is: FFFCh

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

Autumn, 2018

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: 320

The value of the flags field is: FFFDh

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

Autumn, 2018

**test file: ass1mi.out**

Reading file ass1.pls ....Please wait

Printing buffer parameters:

The capacity of the buffer is: 5084

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

The value of the flags field is: FFFCh

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

Autumn, 2018

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: 320

The value of the flags field is: FFFDh

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

Autumn, 2018

**test file: ass1fi.out**

Reading file ass1.pls ....Please wait

The last character read from the file is: t 116

The input file ass1.pls has not been 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

The value of the flags field is: FFFCh

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: 201

The value of the flags field is: FFFDh

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

**test file: 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

The value of the flags field is: FFFCh

Printing buffer contents:

Empty buffer!

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: 1

The value of the flags field is: FFFCh

Printing buffer contents:

