REPORT

과 목 명 : 자료구조
담담교수 : 황두성 교수님
소　　속 : 소프트웨어학과
학　　번 : 32151671
이　　름 : 박민혁

단국대학교
Dankook University

1번.

```
//이프로그램은ull binary tree를전제로하여짜여짐
-1.h-
#pragma warning(disable:4996)
#include <stdio.h>
#include <stdlib.h>

int stack[30];//스택선언
nt top = -1;//스택탑
ypedef struct Linkedlist* Slist
Slist queue[20];//큐선언
nt rear2 = 0, front2 = 0;//큐의레어프론트

ypedef struct Linkedlist
{
char element;
Slist leftchild;
Slist rightchild;
bool isnumsub = false
}Linkedlist//연결리스트선언두자식노드를가짐바이너리트리

-function.c-

include"1.h"
void enqueue(Slist data)
{
if (rear2 + 1 % 20 != front2)
{
rear2 = rear2 + 1 % 20;
queue[rear2] = data
}
}//enqueue과정

list dequeue()
{
if (rear2 != front2)
{
front2 = front2 + 1 % 20;
return queue[front2];
}

return NULL
}//dequeue과정

nt pop()
```

```
{
if (top < 0)
{
printf("stack is empty");
return top;
}
return stack[top--];
}//스택의팝

har push(char tmp)
{
if (sizeof(stack) - 1 == top)
{
printf("stack is full");
return 0;
}
stack[++top] = tmp
}//스택푸시

nt isNum(char tmp)
{
if ('0' <= tmp && tmp <= '9')
return 1;
else
return 0;
}//숫자인지확인해주는함수

nt StackPriority(char tmp)
{
switch (tmp)
{
case '(': return 1;
case ')': return 4;
case '*':
case '/': return 3;
case '+':
case '-': return 2;
}
}//스택우선순위

nt opPriority(char tmp)
{
switch (tmp)
{
case '(':
case ')': return 4;
```

```
case '*':
case '/': return 3;
case '+':
case '-': return 2;
}
}//연산자우선순위

oid convert(char * infix, char * postfix)
{
int cnt = 0;
char tmp;
push(0);

for (int i = 0; infix[i] != 0; i++)
{
tmp = infix[i];
if (isNum(tmp))
postfix[cnt++] = tmp;
else
{
if (tmp == ')')
{
do
{
postfix[cnt++] = pop();
} while (postfix[cnt - 1] != '(');
cnt--;
continue
}
while (StackPriority(stack[top]) >= opPriority(tmp))
{
postfix[cnt++] = pop();
}
push(tmp);
}
}
do {
postfix[cnt++] = pop();
} while (postfix[cnt - 1] != 0);
}//infix를ostfix로변환tree넣기에ostfix가더편하다고생각하여서변환하였습니다)

Slist inittree(char elem)
{
Slist rt = (Slist)malloc(sizeof(Linkedlist));
rt->element = elem
rt->leftchild = NULL
```

```
rt->rightchild = NULL

if (isNum(elem) == 1)
rt->isnumsub = true

return rt;
}//트리생성및초기화

list mergetree(Slist prt, Slist prt2)
{
int cnt = 0;
Slist last[20];

while (1)
{
if (prt2->isnumsub == true)//연결하려는트리가숫자나서브트리이면

f (prt->rightchild == NULL)//오른쪽차일드가비었을때

rt->rightchild = prt2//집어넣음
break
}
else
{
if (prt->rightchild->isnumsub == true)//오른쪽차일드가있고숫자나서브트리이면

f (prt->leftchild == NULL)//또왼쪽차일드가비었을때

rt->leftchild = prt2//왼쪽차일뜨에대입
reak
}
else//아니면왼쪽으로포인터옮김

ast[cnt] = prt
cnt++;
prt = prt->leftchild;

}
}
else//오른쪽에있는데오퍼레이터이면왼쪽으로옮긴다
{
last[cnt] = prt
cnt++;
prt = prt->leftchild;
}
}
```

```
}
else//만약오퍼레이터이면

f (prt->leftchild == NULL)//왼쪽차일드가비었을때왼쪽대입

rt->leftchild = prt2
break
}
else//아니면포인터를왼쪽으로옮긴다
{
last[cnt] = prt
cnt++;
prt = prt->leftchild;
}
}
}

return prt
}//트리합치는알고리즘

list makesubtree(Slist prt, char elem1, char elem2)
{
Linkedlist* node = (Linkedlist*)malloc(sizeof(Linkedlist));
Linkedlist* node2 = (Linkedlist*)malloc(sizeof(Linkedlist));

node->element = elem1
node->leftchild = NULL
node->rightchild = NULL

node2->element = elem2
node2->leftchild = NULL
node2->rightchild = NULL

prt->leftchild = node2;
prt->rightchild = node;
prt->isnumsub = true

return node;
}//서브트리생성

oid inorder(Slist r)
{
if (r->leftchild) inorder(r->leftchild);
printf("%c ", r->element);
if (r->rightchild) inorder(r->rightchild);
}//infix order
```

```
void preorder(Slist r)
{
printf("%c ", r->element);
if (r->leftchild) preorder(r->leftchild);
if (r->rightchild) preorder(r->rightchild);
}//prefix order

void postorder(Slist r)
{
if (r->leftchild) postorder(r->leftchild);
if (r->rightchild) postorder(r->rightchild);
printf("%c ", r->element);
}//postfix order

void levelorder(Slist visit)
{
enqueue(visit);

while (front2 != rear2)
{
visit = dequeue();
printf("%c ", visit->element);
if (visit->leftchild)
enqueue(visit->leftchild);
if (visit->rightchild)
enqueue(visit->rightchild);
}
}//level order

void divtree(int cnt, int tmp, char* postfix, Slist root)
{
int i;

Slist* a = (Slist*)malloc(sizeof(Slist)*cnt);

for (i = cnt - 2; i > -1; i--)
{
if (isNum(postfix[i]) == 0)//스택값이숫자가아니면

f (isNum(postfix[i - 1]) == 1 && isNum(postfix[i - 2]) == 1)//기호옆에두개의스택이숫자면서브
트리로만듬

[tmp] = inittree(postfix[i]);
makesubtree(a[tmp], postfix[i - 1], postfix[i - 2]);
i -= 2;
```

```
tmp++;
}
else//아니면기호만따로만듬

[tmp] = inittree(postfix[i]);
tmp++;
}
}
else//숫자이면숫자만따로만든다
{
a[tmp] = inittree(postfix[i]);
tmp++;
}
}

for (i = 0; i < tmp i++)
{
mergetree(root, a[i]);//트리를다합친다
}
}//트리를나누고합침
```

-main.c-

```
#include"1.h"
int main()
{
char infix[20], postfix[20];
int i, cnt = 0, tmp = 0;
Slist root;

printf("수식을입력하시오 ");
scanf("%s", &infix);

convert(infix, postfix);

for (i = 0; i < 20; i++)
{
if (postfix[i] == NULL)
break

cnt++;
}

root = inittree(postfix[cnt - 1]);

divtree(cnt, tmp, postfix, root);
```

```
printf("infix: ");
inorder(root);
printf("₩n");

printf("prefix: ");
preorder(root);
printf("₩n");

printf("postfix: ");
postorder(root);
printf("₩n");

printf("levelorder: ");
levelorder(root);
printf("₩n");

return 0;
}
```

```
수식을 입력하시오. 6+7x+11*(6x-4)/5
infix: + 6 x 7 - 1 * 1 x 6 / 4 + 5
prefix: + / * - + x 6 7 1 x 1 6 4 5
postfix: 6 7 x + 1 - 1 6 x * 4 / 5 +
levelorder: + / 5 * 4 - x + 1 1 6 x 6 7
```

2번.

-huff.h-

```
#define _CRT_SECURE_NO_WARNINGS
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<string.h>

#define ALPHA 256 //ALPHANUM
#define BUFSIZE 1024 //버퍼사이즈

ypedef struct _huffNode
{
int freq;
char c;
```

```
struct _huffNode *left, *right;

}huffNode //호프만트리를위한노드

truct _huffNode **heap;
int lastHeapIdx = 0;
char codeBuf[100];
int codeBufIdx = -1;
int freq[ALPHA];
char *SYM_CODE[ALPHA];
void showFreq(); //빈도수출력
oid addToHeap(huffNode *cur);
huffNode* deleteFromHeap();
void through(huffNode *cur, char c);
int countNonZeroC();
void Decoding();
void justthrough(huffNode *cur, char c);
```

-main.c-

```
#define _CRT_SECURE_NO_WARNINGS
#include "huff.h"

int main(int argc, char* argv[])
{
char filein[100];
char fileout[100];
unsigned char buff[1];
int i, j;
int k = 0;
FILE *inputFile;
char ch;
char buf[BUFSIZE];

if (argc == 3){
strcpy(filein, argv[0]);
strcpy(fileout, argv[1]);
}
else{
printf("incoding file input : ₩n");
gets_s(filein);
printf("incoding file output : ₩n");
gets_s(fileout);
}

if ((inputFile = fopen(filein, "rb")) == NULL){
```

```
printf("Unable to open %s for input₩n", filein);
exit(1);
}
// 파일오픈
emset(freq, 0, ALPHA);
while (fread(buff, sizeof(char), 1, inputFile))
{
freq[(int)buff[0]]++;
}
fclose(inputFile);
printf("₩n------ 문자빈도수----- ₩n");
showFreq(); //빈도수출력
rintf("₩n------------------------- ₩n");

int cnt = countNonZeroC();//빈도수가이상인것카운트
eap = (huffNode **)malloc((cnt + 1) * sizeof(huffNode *));//힙공간생성
emset(heap, 0, (cnt + 1) * sizeof(huffNode *));//minHeap 구성

or (int i = 0; i < ALPHA i++)
{
if (freq[i] > 0)
{
huffNode *current = (huffNode *)malloc(sizeof(huffNode));
current->c = (char)i;
current->freq = freq[i];
current->left = current->right = 0;
addToHeap(current);
}

}//huffman Tree 구성

uffNode *first = 0;
huffNode *second = 0;

while (1)
{
first = deleteFromHeap();
second = deleteFromHeap();

if (second == 0)
{
printf("₩n호프만트리완료");
break
}
huffNode *newOne = (huffNode*)malloc(sizeof(huffNode));
newOne->c = 0;
```

```
newOne->freq = first->freq + second->freq;
newOne->left = first;
newOne->right = second;
addToHeap(newOne);

}//first는허프만트리의루트노드를가리킴
emset(SYM_CODE, 0, sizeof(SYM_CODE));
through(first->left, '0');
through(first->right, '1');
//////////////////////////////////////////////////////
int numofSym = 0;
printf("₩n-------매칭결과-----₩n");
for (int i = 0; i < ALPHA i++)
{if (SYM_CODE[i] != 0)
{
numofSym++;
printf("symbol : %c =>> %s ₩n", (char)i, SYM_CODE[i]);
}
}
printf("₩n----------------------₩n");

FILE *fout = fopen(fileout, "wb");
if (fout != 0)
{
fwrite(&numofSym, sizeof(numofSym), 1, fout);
char write_BUF[100];
for (int i = 0; i < ALPHA i++)
{
if (SYM_CODE[i] != 0)
{
write_BUF[0] = (char)i;
write_BUF[1] = (char)strlen(SYM_CODE[i]);
strcpy(&write_BUF[2], SYM_CODE[i]);
fwrite(write_BUF, sizeof(char), 2 + strlen(SYM_CODE[i]), fout);
}
}
}
else
printf("error...₩n");

FILE *inFile = fopen(filein, "rt");

if (inFile != 0)
{
int locTotalNumBit;
locTotalNumBit = ftell(fout);
```

```
if (fseek(fout, 4, SEEK_CUR) != 0)
{
printf("Failed to move the file pointer ₩n");

fclose(inFile);
fclose(fout);
return 0;
}
//실제로파일을읽어서인코딩

har bitbuf[BUFSIZE]; //비트스트림저장버퍼
nt bitshiftCnt = 7;//비트시프트횟수
nt bitBufIdx = 0; //비트스트림저장위치
nt totalBitNum = 0;
//실제로파일을읽어서인코딩
har flag = 0;//flag가이면기록할것없음
/bit buf를으로초기화
emset(bitbuf, 0, BUFSIZE);

while (fgets(buf, BUFSIZE, inFile) != 0)
{
int len = strlen(buf);

for (int i = 0; i < len; i++)
{
char *hufmanCode = SYM_CODE[(int)buf[i]];

for (unsigned int j = 0; j < strlen(hufmanCode); j++)
{
char val = 0;
if (hufmanCode[j] == '0')
{
val = 0;
}
else if (hufmanCode[j] == '1')
{
val = 1;
}
else
{
printf("error...");
}

val = val << bitshiftCnt;
bitshiftCnt--;
```

```
bitbuf[bitBufIdx] |= val;

flag = 1;
totalBitNum++;


if (bitshiftCnt < 0)
{
bitshiftCnt = 7;
bitBufIdx++;

if (bitBufIdx >= BUFSIZE)
{
fwrite(bitbuf, 1, BUFSIZE, fout);
flag = 0;
bitBufIdx = 0;
memset(bitbuf, 0, BUFSIZE);
}
}
}
}
}

if (flag == 1)
{
fwrite(bitbuf, 1, bitBufIdx + 1, fout);
}

if (fseek(fout, locTotalNumBit, SEEK_SET) == 0)
{
fwrite(&totalBitNum, sizeof(totalBitNum), 1, fout);
}
else
{
printf("unable to record total number of bit₩n");
}


fclose(inFile);
}
else
{
printf("Unable File to open ");
}
fclose(fout);
//전체비트갯수기록할위치기억해둠
```

```
ecoding();
//Encodeing file contents
}
```

-function.c-

```
#include"huff.h"

void addToHeap(huffNode *cur)
{
lastHeapIdx++;
heap[lastHeapIdx] = cur

int currentIdx = lastHeapIdx;
int parentIdx = currentIdx / 2;

while (parentIdx >= 1)
{
if (heap[parentIdx]->freq > heap[currentIdx]->freq)
{
//부모와나를바꿈
uffNode *temp = heap[parentIdx];
heap[parentIdx] = heap[currentIdx];
heap[currentIdx] = temp;

currentIdx = parentIdx;
parentIdx = currentIdx / 2;
}
else
break

}

}

huffNode* deleteFromHeap()
{

if (lastHeapIdx <= 0)
{
printf("₩nheap is empty ");
return 0;
}

huffNode *returnVal = heap[1];
heap[1] = heap[lastHeapIdx];
```

```
lastHeapIdx--;

int parent = 1;
int left = 2 * parent;
int right = left + 1;

while (1)
{
if (left > lastHeapIdx) //자식이없을때

reak
}
else if (right > lastHeapIdx) // 왼쪽자식만가짐

f (heap[left]->freq < heap[parent]->freq)
{
huffNode *temp = heap[left];
heap[left] = heap[parent];
heap[parent] = temp;

parent = left;
left = 2 * parent;
right = left + 1;
}
else
{
break
}
}
else //양쪽다자식이있을때

nt smaller;

if (heap[left]->freq <= heap[right]->freq)
{
smaller = left;
}
else
{
smaller = right;
}

if (heap[smaller]->freq < heap[parent]->freq)
{
huffNode *temp = heap[smaller];
heap[smaller] = heap[parent];
```

```
heap[parent] = temp;

parent = smaller;
left = 2 * parent;
right = left + 1;
}
else
{
break
}
}
}

return returnVal;
}

void justthrough(huffNode *cur, char c)
{
codeBufIdx++;
codeBuf[codeBufIdx] = c
codeBuf[codeBufIdx + 1] = 0;

if (cur->left == 0 && cur->right == 0)
{
printf("charcter %d (%c) : %s ₩n", (int)cur->c, cur->c, codeBuf);
}
else
{
justthrough(cur->left, '0');
justthrough(cur->right, '1');
}
codeBuf[codeBufIdx] = 0;
codeBufIdx--;
return


}

void through(huffNode *cur, char c)
{
codeBufIdx++;
codeBuf[codeBufIdx] = c
codeBuf[codeBufIdx + 1] = 0;

if (cur->left == 0 && cur->right == 0)
{
```

```
//printf("charcter %d (%c) : %s ₩n", (int)cur->c, cur->c, codeBuf);
char* hufCode = (char*)malloc(strlen(codeBuf) + 1);
strcpy(hufCode, codeBuf);
SYM_CODE[(int)cur->c] = hufCode;

}
else
{
through(cur->left, '0');
through(cur->right, '1');
}

codeBuf[codeBufIdx] = 0;
codeBufIdx--;
return


}

//출현빈도수가이상인문자들의총개수
nt countNonZeroC()
{
int cnt = 0;
for (int i = 0; i < ALPHA i++)
{
if (freq[i] > 0)
{
cnt++;
}
}
return cnt;
}

void showFreq()
{
int i;

for (i = 0; i < 256; i++)
{
if (freq[i] == 0)
continue
printf("%c => %d ₩n", i, freq[i]);

}
}// 빈도수출력
```

```
oid Decoding()
{

char filein[100];
char fileout[100];
printf("decoding file input : ₩n");
gets_s(filein);
printf("decoding file output : ₩n");
gets_s(fileout);

FILE *infile = fopen(filein, "rb");
FILE *outfile = fopen(fileout, "wb");

if (infile != 0)
{
//허프만트리재구성
nt numOfSym = 0;
fread(&numOfSym, sizeof(int), 1, infile);

printf("number of symbol %d ₩n", numOfSym);


huffNode *huffRoot = (huffNode *)malloc(sizeof(huffNode));
huffRoot->left = huffRoot->right = 0;
huffNode *cur = huffRoot;



for (int i = 0; i < numOfSym; i++)
{

char symbolAndLen[2]; // 0: symbol , 1: 길이
read(symbolAndLen, 2, 1, infile);
char buf[BUFSIZE];
fread(buf, 1, (int)symbolAndLen[1], infile);
buf[(int)symbolAndLen[1]] = 0;
printf("%c : (%d) ---- %s ₩n", symbolAndLen[0], (int)symbolAndLen[1], buf);
cur = huffRoot;


for (int j = 0; j < (int)symbolAndLen[1]; j++)
{
if (buf[j] == '0')
{
if (cur->left == 0)
{
```

```
cur->left = (huffNode*)malloc(sizeof(huffNode));
cur->left->left = 0;
cur->left->right = 0;
}
cur = cur->left;

}
else if (buf[j] == '1')
{
if (cur->right == 0)
{
cur->right = (huffNode*)malloc(sizeof(huffNode));
cur->right->left = 0;
cur->right->right = 0;
}
cur = cur->right;
}
else
{
printf("error...");
exit(0);
}
}
cur->c = symbolAndLen[0];
}

//허프만트리복구완료
//디코딩수행
/codeBufIdx = -1;
//justthrough(huffRoot->left,'0');
//justthrough(huffRoot->right, '1');

int numBitsToread = 0;
fread(&numBitsToread, sizeof(int), 1, infile);
printf("total bits %d₩n₩n", numBitsToread);

cur = huffRoot;
char buf[BUFSIZE];
while (1)
{
int sz = fread(buf, 1, BUFSIZE, infile);
if (sz == 0)
{
printf("End of file ₩n");
break
}
```

```
else
{
for (int i = 0; i < sz; i++)
{
for (int j = 0; j < 8; j++)
{
if ((char)(buf[i] & 0x80) == 0)
{
cur = cur->left;
}
else
{
cur = cur->right;
}

buf[i] = buf[i] << 1;
numBitsToread--;

if (cur->left == 0 && cur->right == 0)
{
printf("%c", cur->c);
fprintf(outfile, "%c", cur->c);
cur = huffRoot;
}
if (numBitsToread == 0)
{
printf("End of Decoding₩n");
return
}
}
}
}
}
fclose(outfile);
fclose(infile);
}
else
printf("error..₩n");
}
```

```
0 => 9
1 => 5
2 => 2
3 => 2
4 => 4
5 => 2
7 => 1
8 => 3
9 => 6
A => 5
B => 1
C => 4
H => 8
I => 7
K => 9
L => 1
M => 1
N => 9
O => 1
P => 3
R => 1
S => 7
T => 7
a => 222
b => 31
c => 83
d => 109
e => 372
f => 77
g => 66
h => 130
i => 203
j => 3
k => 25
l => 123
m => 75
n => 229
o => 247
p => 59
q => 2
r => 208
s => 215
t => 255
u => 77
v => 39
w => 46
x => 6
y => 61
```

```
heap is empty
호프만 트리 완료

-------매칭 결과------
symbol :
 =>> 0100001
 =>> 0100000
symbol :    =>> 111
symbol : 0 =>> 110100011
symbol : 1 =>> 000010110
symbol : 2 =>> 11010000011
symbol : 3 =>> 0000101000
symbol : 4 =>> 11010000000
symbol : 5 =>> 11010000100
symbol : 7 =>> 110100000100
symbol : 8 =>> 0100010100
symbol : 9 =>> 010001001
symbol : A =>> 000010101
symbol : B =>> 110000011010
symbol : C =>> 1100000111
symbol : H =>> 110000010
symbol : I =>> 010001011
symbol : K =>> 110100010
symbol : L =>> 110100000101
symbol : M =>> 110000011000
symbol : N =>> 00001000
symbol : O =>> 110000011001
symbol : P =>> 0100010101
symbol : R =>> 110000011011
symbol : S =>> 110000000
symbol : T =>> 110000001
symbol : a =>> 0111
symbol : b =>> 1100001
symbol : c =>> 00000
symbol : d =>> 01001
symbol : e =>> 001
symbol : f =>> 110110
symbol : g =>> 110001
symbol : h =>> 11001
symbol : i =>> 0001
symbol : j =>> 0000101001
symbol : k =>> 0100011
symbol : l =>> 10011
symbol : m =>> 110101
symbol : n =>> 1000
symbol : o =>> 1010
symbol : p =>> 100100
symbol : q =>> 11010000101
symbol : r =>> 0101
symbol : s =>> 0110
symbol : t =>> 1011
symbol : u =>> 110111
symbol : v =>> 1101001
symbol : w =>> 000011
symbol : x =>> 000010111
symbol : y =>> 100101
```

makefile.

```
all : main
main : main.o functions.o
        gcc -o main main -o functions,o
main.o : main.c functions.h
        gcc -c main.c
functions.o : functions.c functions..h
        gcc -c functions.c
```