<<8. 9과제>>

과제 1. sin 그래프 별로 찍어보기 (실행파일 첨부)

과제 2. unsigned char 변수의 값을 비트단위로 찍어주기

- 지역변수에 특정 값을 하나 넣음

- 비트 단위로 분해해서 0 이면 0 출력, 1 이면 1 출력

과제 3. unsigned short (16bit) 변수의 각 비트를 컨트롤 하기

- unsigned short 변수 = 0 으로 초기값 가짐.

- 키보드로 1 ~ 16의 비트 자리 입력을 받음

- 1 / 0 을 사용자로부터 받아서 지정된 자리의 비트를 0 또는 1로 바꿔줌.

- 다른 위체이 입력된 기존 데이터는 보존이 되어야 함

과제 4. unsigned int (32bit) 변수를 바이트 단위로 사용하기

- unsigned int 변수 = 0 초기값 가짐

- 키보드로 1 ~ 4 의 바이트 위치를 입력 받고

- 해당 위치에 넣을 데이터 0 ~ 255 를 입력 받음

- 사용자가 입력한 바이트 위치에 해당 값을 넣음

- 데이터가 입력 되면 바이트 단위로 쪼개서 출력 & 4바이트 16진수 출력

- 기존 데이터는 보존이 되어야 하며

- 입력된 바이트 위치의 데이터는 기존 값을 지우고 넣음.

#include "stdafx.h"

/\*

과제 1. sin 그래프 별로 찍어보기 (실행파일 첨부)

\*/

0~ 180도 로 해볼것

몇번을 반복시킬것이냐를 사인이나 코사인값으로 쓰는 것

double GetRadian(double degree)

{

return degree \* 0.0174533;

}

void SinGraph()

{

int i, j;

while (true)

{

for (i = 0; i <= 360; i += 7)

{

for (j = -40; j < sin(GetRadian(i)) \* 45; j++)

{

printf("\*");

}

printf("\n"); //그래프 찍고 줄바꿈

Sleep(15);

}

}

}

void UCtoB()

{

/\*

과제 2. unsigned char 변수의 값을 비트단위로 찍어주기

- 지역변수에 특정 값을 하나 넣음

- 비트 단위로 분해해서 0 이면 0 출력, 1 이면 1 출력

\*/

unsigned char Input;

int i;

while (true)

{

printf("Input Character : ");

scanf\_s("%d", &Input);

for (i = sizeof(Input) \* 8 - 1; i >= 0; --i)

{

int mask = 1 << i;

if (Input & mask)

cout << 1;

else cout << 0;

if (i % 4 == 0)

printf(" ");

}

cout << endl << endl;

}

}

void BitInput()

{

/\*

Shift, or, and의 활용

과제 3. unsigned short (16bit) 변수의 각 비트를 컨트롤 하기

- unsigned short 변수 = 0 으로 초기값 가짐.

- 키보드로 1 ~ 16의 비트 자리 입력을 받음

- 1 / 0 을 사용자로부터 받아서 지정된 자리의 비트를 0 또는 1로 바꿔줌.

- 다른 위체이 입력된 기존 데이터는 보존이 되어야 함

\*/

unsigned short Init = 0;

while (true)

{

int Input;

int Switch;

system("cls");

printf("비트위치 : ");

scanf\_s("%d", &Input);

int Digit = 1 << (Input - 1);

printf("\nOFF/ON [0,1] :");

scanf\_s("%d", &Switch);

//비트가 바뀌는부분

if (Switch == true)

Init |= Digit;

else if (Switch == false)

Init &= (~Digit);

cout << "값 : " << Init << endl;

//출력

int i;

for (i = sizeof(Init) \* 8 - 1; i >= 0; --i)

{

int mask = 1 << i;

printf("%d번 Bit : ", i + 1);

if (mask & Init)

printf("ON");

else printf("OFF");

cout << endl;

}

cout << endl;

system("pause");

}

}

/\*

과제 4. unsigned int (32bit) 변수를 바이트 단위로 사용하기

- unsigned int 변수 = 0 초기값 가짐

- 키보드로 1 ~ 4 의 바이트 위치를 입력 받고

- 해당 위치에 넣을 데이터 0 ~ 255 를 입력 받음

- 사용자가 입력한 바이트 위치에 해당 값을 넣음

- 데이터가 입력 되면 바이트 단위로 쪼개서 출력 & 4바이트 16진수 출력

- 기존 데이터는 보존이 되어야 하며

- 입력된 바이트 위치의 데이터는 기존 값을 지우고 넣음.

\*/

void PrintByte()

{

int Init = 0;

int Location;

int Loc\_Val[4] = { 0, };

int Total\_Val = 0;

int Digit;

int i;

while (true)

{

printf("위치 (1~4) : ");

scanf\_s("%d", &Location);

printf("값 [0~255] : ");

scanf\_s("%d", &Loc\_Val[Location - 1]);

//범위 유효성검사

if (Location < 1 || Location > 4

|| Loc\_Val[Location - 1] < 0

|| Loc\_Val[Location - 1] > 255)

{

printf("위치 범위를 초과하였습니다.\n\n");

continue;

}

for (i = 0; i < \_countof(Loc\_Val); ++i)

{

printf("%d 번째 바이트 값 : %d\n", i + 1, Loc\_Val[i]);

}

//0000 0000 0000 0000 0000 0000 0000 0000

/\*

1 = 1

2 = 9

3 = 17

4 = 25

(x-1)\*8 + 1 - 1

\*/

Digit = 1 << (Location - 1) \* 8;

Digit \*= Loc\_Val[Location - 1];

Total\_Val += Digit;

printf("전체 4바이트 값 : 0x%08X\n\n", Total\_Val);

}

}

void PrintByte()

{

unsigned int total = 0;

int bit\_value[4] = { 0, };

int byte\_location;

while (true)

{

printf("위치 (1~4) : ");

scanf\_s("%d", &byte\_location);

printf("값 [0~255] : ");

scanf\_s("%d", &bit\_value[byte\_location - 1]);

int i;

for (i = 0; i < \_countof(bit\_value); ++i)

{

printf("%d번째 바이트 값 : %d", i + 1, bit\_value[i]);

cout << endl;

}

printf("전체 4바이트 값 0x : ");

for (i = \_countof(bit\_value); i > 0; --i)

{

printf("%02X", bit\_value[i - 1]);

}

putchar('\n');

}

}

**<<8.12 과제>>**

**재귀를 활용하여 페인트 알고리즘을 구현**

void Recursive\_Paint(int arr[][15], const int col, const int row,

int Dstcol, int Dstrow)

{

if (Dstrow < 0 || Dstrow >= row || Dstcol < 0 || Dstcol >= col)

return;

system("cls");

arr[Dstrow][Dstcol] = '.';

int i, j;

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

{

for (j = 0; j < col; ++j)

{

if (arr[i][j] == 0)

printf("%d ", arr[i][j]);

else if(arr[i][j] == '.' || arr[i][j] == ' ')

printf("%c ", arr[i][j]);

}

printf("\n");

}

if (arr[Dstrow - 1][Dstcol] == 0)

Recursive\_Paint(arr, col, row, Dstcol, Dstrow - 1);

if (arr[Dstrow + 1][Dstcol] == 0)

Recursive\_Paint(arr, col, row, Dstcol, Dstrow + 1);

if (arr[Dstrow][Dstcol - 1] == 0)

Recursive\_Paint(arr, col, row, Dstcol - 1, Dstrow);

if (arr[Dstrow][Dstcol + 1] == 0)

Recursive\_Paint(arr, col, row, Dstcol + 1, Dstrow);

}

int \_tmain(int argc, TCHAR\*\* argv)

{

int arr[10][15] = {

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ' ', ' ', ' ', 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ' ', 0, ' ', 0, 0,

0, 0, ' ', ' ', 0, 0, 0, 0, 0, ' ', 0, 0, ' ', 0, 0,

0, 0, ' ' , 0, ' ', ' ', 0, 0, 0, ' ', ' ', ' ', ' ', 0, 0,

0, 0, 0, ' ', ' ' , ' ', 0, 0, ' ', ' ', ' ', ' ', ' ', 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ' ', ' ', 0, 0,

' ', ' ', 0, 0, ' ', ' ', 0, 0, 0, 0, 0, ' ', ' ', 0, 0,

0, 0, 0, ' ', 0, 0, ' ', 0, 0, 0, 0, ' ', ' ', 0, 0,

0, 0, 0, ' ', 0, 0, ' ', 0, 0, 0, 0, ' ', ' ', 0, 0,

};

int col = sizeof(arr[0]) / sizeof(arr[0][0]);

int row = sizeof(arr) / sizeof(arr[0]);

Recursive\_Paint(arr, col, row, 0, 0);

return 0;

}

**<<8.14 과제>>**

**재귀x 스택을 사용한 페인트 알고리즘**

**스택헤더**

#pragma once

#ifndef \_\_CMYSTACK\_H\_\_

#define \_\_CMYSTACK\_H\_\_

typedef struct Data

{

Data() {}

Data(int \_x, int \_y) : x(\_x), y(\_y) {}

int x;

int y;

};

struct Node

{

Node() {};

Node(Data \_data) : data(\_data) {}

Data data;

Node \*next = nullptr;

};

class CMyStack

{

public:

explicit CMyStack();

virtual ~CMyStack();

public:

void Initalize();

public:

bool ISEmpty();

void Push(Data data);

bool Pop(Data& data);

bool Seek(Data& data);

private:

Node headNode;

};

#endif // !\_\_CMYSTACK\_H\_\_

**스택 CPP**

#include "stdafx.h"

#include "CMyStack.h"

CMyStack::CMyStack()

{

Initalize();

}

CMyStack::~CMyStack()

{

}

void CMyStack::Initalize()

{

headNode.data.x = 0;

headNode.data.y = 0;

headNode.next = nullptr;

}

bool CMyStack::ISEmpty()

{

if (headNode.next == nullptr)

return true;

else

return false;

}

void CMyStack::Push(Data data)

{

Node\* NewNode = new Node(data);

NewNode->next = headNode.next;

headNode.next = NewNode;

}

bool CMyStack::Pop(Data & data)

{

if (ISEmpty())

return false;

Node\* PopNode = headNode.next;

data = PopNode->data;

headNode.next = PopNode->next;

delete[] PopNode;

PopNode = nullptr;

return true;

}

bool CMyStack::Seek(Data& data)

{

if (ISEmpty())

return false;

data = headNode.next->data;

return true;

}

**Main 분실;**

**<8.19 과제>**

**QuickSort의 구현**

**<8.21 과제>**

**번역기 구현**

// assignment.cpp: 콘솔 응용 프로그램의 진입점을 정의합니다.

//

#include "pch.h"

int \_tmain(int argc, TCHAR\*\* argv)

{

char Input[128];

char\* tmp[64] = { 0, };

const char\* Deli = " ";

char \*Context = '\0';

char\* tok;

int i = 0;

gets\_s(Input, sizeof(Input));

tok = strtok\_s(Input, Deli, &Context);

tmp[i] = tok;

while (true)

{

tok = strtok\_s(Context, Deli, &Context);

tmp[++i] = tok;

if (tok == '\0')

break;

}

char Table[10][2][20] =

{

{ "HELLO", "안녕" },

{ "WORLD", "세계" },

{ "VISUAL", "비주얼" },

{ "STUDIO", "스튜디오" },

{ "i", "나는" },

{ "boy", "소년" },

{ "add", "더하기" },

{ "subtract", "빼기" },

{ "multiply", "곱하기" },

{ "division", "나누기" }

};

int Index\_1 = sizeof(Table) / sizeof(Table[0]);

int Index\_2 = sizeof(Table[0]) / sizeof(Table[0][0]);

int Index\_3 = sizeof(Table[0][0]) / sizeof(Table[0][0][0]);

int j;

int k = 0;

for (i = 0; tmp[i] != '\0'; ++i)

{

for (j = 0; j < Index\_1; ++j)

{

if (strcmp(Table[j][0], tmp[i]) == 0)

{

tmp[i] = Table[j][1];

printf("%s ", tmp[i]);

}

}

}

return 0;

}

**<8.21 과제 추가 >**

**손코딩**

**프린트물에 나와있는 함수.**

**+ strupr (문자열을 대문자로 변환), strlwr(문자열을 소문자로 변환)**

**<8.23 과제>**

**시간맞추기**

// Project1.cpp : 이 파일에는 'main' 함수가 포함됩니다. 거기서 프로그램 실행이 시작되고 종료됩니다.

//

#include "pch.h"

struct Result

{

double click = 0;

char score[12] = { 0, };

};

int main()

{

int timing[] = { 5, 10, 14, 17, 20, 25, 29, 31, 33 };

Result res[\_countof(timing)];

int Index = 0;

while (true)

{

system("cls");

//시간출력

double time = clock() / 1000.0;

printf("\t [ %0.3lf ]\n\n\n", time);

//점수판출력

for (int i = 0; i < \_countof(timing); ++i)

printf("%d Sec : %s\n", timing[i], res[i].score);

//게임종료

if (res[\_countof(res) - 1].click != '\0'

|| time > timing[\_countof(timing) - 1] + 2)

break;

////////////////////////////////////////////////////////////////

//입력확인

if (\_kbhit())

{

res[Index].click = time;

if (abs(timing[Index] - res[Index].click) <= 0.10)

strcpy\_s(res[Index].score, "Perfect");

else if (abs(timing[Index] - res[Index].click) <= 0.30)

strcpy\_s(res[Index].score, "Great");

else if (abs(timing[Index] - res[Index].click) <= 0.50)

strcpy\_s(res[Index].score, "Good");

else

strcpy\_s(res[Index].score, "fail");

if (Index < \_countof(res) - 1)

++Index;

\_getch();

}

//시간초과

if (timing[Index] + 0.5 <= time)

{

strcpy\_s(res[Index].score, "fail");

if (Index < \_countof(res) - 1)

++Index;

}

}

}

**<8.26 과제>**

**큐 커멘드센터**

**Queue.h**

#pragma once

#ifndef \_\_QUEUE\_H\_\_

#define \_\_QUEUE\_H\_\_

struct Unit

{

Unit() {};

int InitTime;

};

typedef Unit Data;

typedef struct \_node

{

Data data;

struct \_node \* next = nullptr;

}Node;

typedef struct \_lQueue

{

Node \*front = nullptr;

Node \*rear = nullptr;

}LQueue;

typedef LQueue Queue;

void QueueInit(Queue\* pq);

int QIsEmpty(Queue\* pq);

void Enqueue(Queue\* pq, Data data);

bool Dequeue(Queue \* pq, Data\* data);

bool QPeek(Queue \* pq, int Index, Data\* res);

#endif

**Queue.cpp**

#include "stdafx.h"

#include "Queue.h"

void QueueInit(Queue \* pq)

{

pq->front = nullptr;

pq->rear = nullptr;

}

int QIsEmpty(Queue \* pq)

{

return (pq->front == nullptr) ||

(pq->rear == nullptr);

}

void Enqueue(Queue \* pq, Data data)

{

Node\* NewNode = new Node;

NewNode->data = data;

NewNode->next = nullptr;

if (QIsEmpty(pq))

{

pq->front = NewNode;

pq->rear = NewNode;

}

else

{

pq->rear->next = NewNode;

pq->rear = NewNode;

}

}

bool Dequeue(Queue \* pq, Data\* data)

{

if (QIsEmpty(pq))

{

cout << "Under Flow" << endl;

return false;

}

Node\* rPos = pq->front;

\*data = rPos->data;

pq->front = pq->front->next;

delete rPos; rPos = nullptr;

return true;

}

bool QPeek(Queue \* pq, int Index, Data\* res)

{

if (QIsEmpty(pq))

return false;

Node\* pNode = pq->front;

for (int i = 1; i < Index; ++i)

{

pNode = pNode->next;

}

\*res = pNode->data;

return true;

}

**Main**

// assignment.cpp: 콘솔 응용 프로그램의 진입점을 정의합니다.

#include "stdafx.h"

#include "Queue.h"

int \_tmain(int argc, TCHAR\*\* argv)

{

LQueue q;

Unit unit[4];

QueueInit(&q);

int StartCount = GetTickCount64();

int CurrentCount;

int qSize = 0;

while (true)

{

system("cls");

printf("[ 1 : 유닛 생성 요청 / 2 : 종료 ]\n\n");

printf("-- CONTROL TOWER UNIT CREATE --\n");

printf("#-----------------------------------------------------------\n");

for (int i = 1; i <= qSize; ++i)

{

Unit utmp;

QPeek(&q, i, &utmp);

int ttmp = (GetTickCount64() - utmp.InitTime) / 100;

if (ttmp <= 100)

printf("# Progress:%d%% ", ttmp);

else if (ttmp > 100)

{

Dequeue(&q, &Data());

--qSize;

}

}

printf("\n#-----------------------------------------------------------\n");

//////////////////////////////////////////////////////////////////////////////

if (\_kbhit())

{

int Input = \_getch();

if (Input == '1')

{

if (qSize < 4)

{

unit[qSize].InitTime = GetTickCount64();

Enqueue(&q, unit[qSize]);

++qSize;

}

else

printf("\n!! 생성요청 불가 !!");

}

else if (Input == '2')

{

exit('0');

}

}

Sleep(100);

}

}

**<8.28 과제>**

**파일패킹, 해제하기.**

#include "pch.h"

#define CONFIRM 6659

#define EXIT(a) exit(a);

struct PackHeader

{

unsigned long Identifier = 6666;

unsigned int FileCount;

};

struct FileHeader

{

char FileName[64];

int FileSize = 0;

int Offset;

};

int \_tmain(int argc, TCHAR\*\* argv)

{

system("cls");

int Menu\_Select;

printf("[1] 패킹하기 [2] 패킹풀기\n");

scanf\_s("%d", &Menu\_Select);

//패킹하기 선택

if (Menu\_Select == 1)

{

system("cls");

PackHeader ph;

ph.Identifier = CONFIRM;

printf("패킹 할 파일 개수는? ");

scanf\_s("%d", &ph.FileCount);

FileHeader\* fh = new FileHeader[ph.FileCount];

int Total\_HeaderSize = sizeof(PackHeader) + (sizeof(FileHeader) \* ph.FileCount);

//==========================================

// 패킹파일에 들어갈 파일헤더 셋팅

//==========================================

printf("\n파일이름에 확장자 명을 포함합니다.\n");

for (int i = 0; i < ph.FileCount; ++i)

{

printf("%d번째 파일이름 : ", i + 1);

scanf\_s("%s", fh[i].FileName, sizeof(fh[i].FileName));

FILE\* fp;

errno\_t err = fopen\_s(&fp, fh[i].FileName, "rb");

if (err != NULL)

{

printf("ERROR");

EXIT(0);

}

//사이즈

fseek(fp, 0, SEEK\_END);

fh[i].FileSize = ftell(fp);

fseek(fp, 0, SEEK\_SET);

std::fclose(fp);

//오프셋

fh[i].Offset = Total\_HeaderSize;

for (int j = 0; j < i; ++j)

{

fh[i].Offset += fh[j].FileSize;

}

//printf("%d의 위치 : %d\n", i + 1, fh[i].Offset);

}

//===========================================

// 패킹파일 생성 및 열기

//===========================================

char Packing\_FileName[64];

printf("\n패킹될 파일의 이름은? ( 영문64자 내로입력 ) : ");

scanf\_s("%s", Packing\_FileName, sizeof(Packing\_FileName));

FILE\* fp;

errno\_t err = fopen\_s(&fp, Packing\_FileName, "wb");

if (err != NULL)

printf("[ 패킹파일 생성 실패 ]");

//===========================================

// 파일패킹 - 헤더읽어오기

//===========================================

int res = fwrite(&ph, sizeof(PackHeader), 1, fp);

if (res == true)

{

for (int i = 0; i < ph.FileCount; ++i)

{

//헤더 읽어오기

res = fwrite(&fh[i], sizeof(FileHeader), 1, fp);

if (res != true)

printf("\n[ 파일헤더 쓰기실패 ]");

}

}

else

printf("\n[ 패킹헤더 쓰기실패 ]");

//===========================================

// 파일패킹 - 데이터읽어와서 쓰기

//===========================================

for (int i = 0; i < ph.FileCount; ++i)

{

FILE\* fp2;

errno\_t err2 = fopen\_s(&fp2, fh[i].FileName, "rb");

printf("\n %d / %d", i + 1, ph.FileCount);

if (err2 != NULL)

{

printf("\n[ 열기 실패 ]");

EXIT(0);

}

else

printf("\n[ 열기 성공 ]");

//데이터를 파일로부터 읽어오기

char\* tmp = new char[fh[i].FileSize];

int res = fread(tmp, fh[i].FileSize, 1, fp2);

if (res == true)

printf("\n[ 읽기 성공 ]");

else

{

printf("\n[ 읽기 실패 ]");

EXIT(0);

}

//데이터를 패킹파일에 쓰기

res = fwrite(tmp, fh[i].FileSize, 1, fp);

if (res == true)

printf("\n[ 쓰기 성공 ]");

else

{

printf("\n[ 쓰기 실패 ]");

EXIT(0);

}

printf("\n\n");

delete[] tmp;

std::fclose(fp2);

}

delete[] fh;

std::fclose(fp);

printf("[ 패킹 완료 ]\n\n");

}

//패킹풀기

else if (Menu\_Select == 2)

{

system("cls");

char PackFile[64];

printf("분리할 패킹파일 이름은? (64자) : ");

scanf\_s("%s", PackFile, sizeof(PackFile));

FILE\* fp;

errno\_t err = fopen\_s(&fp, PackFile, "rb");

if (err != NULL)

printf("[ 패킹파일 열기 실패 ]");

//==========================================

// 패킹파일에서 패킹헤더 읽어와서 확인

//==========================================

PackHeader \_\_ph;

int res = fread(&\_\_ph, sizeof(PackHeader), 1, fp);

if (res != 1)

{

printf("[ 패킹파일 읽기 실패 ]");

EXIT(0);

}

if (\_\_ph.Identifier != CONFIRM)

{

printf("[ 패킹파일 식별 실패 ]");

EXIT(0);

}

//==========================================

// 패킹파일에서 파일헤더 읽어와서 확인

//==========================================

FileHeader\* \_\_fh = new FileHeader[\_\_ph.FileCount];

res = fread(\_\_fh, sizeof(FileHeader) \* \_\_ph.FileCount, 1, fp);

if (res != 1)

{

printf("[ 헤더파일 읽기 실패 ]");

EXIT(0);

}

printf("\n[ 패킹된 파일 목록 ]\n\n");

printf("0.모든파일\n");

for (int i = 0; i < \_\_ph.FileCount; ++i)

{

printf("%s\n", \_\_fh[i].FileName);

}

printf("\n");

//==========================================

// 패킹파일에서 선택된 파일만 생성

//==========================================

int File\_Select;

scanf\_s("%d", &File\_Select);

if (File\_Select > 0 && File\_Select <= \_\_ph.FileCount)

{

char\* tmp = new char[\_\_fh[File\_Select - 1].FileSize];

fseek(fp, \_\_fh[File\_Select - 1].Offset, SEEK\_SET);

int res = fread(tmp, \_\_fh[File\_Select - 1].FileSize, 1, fp);

if (res != true)

{

printf("[ 패킹파일 읽기 실패 ]");

EXIT(0);

}

FILE\* fp2;

errno\_t err2 = fopen\_s(&fp2, \_\_fh[File\_Select - 1].FileName, "wb");

if (err2 != NULL)

{

printf("[ %s 파일생성 실패 ]", \_\_fh[File\_Select - 1].FileName); //여기서 오류

EXIT(0);

}

res = fwrite(tmp, \_\_fh[File\_Select - 1].FileSize, 1, fp2);

if (res != true)

{

printf("[ %s 파일쓰기 실패 ]", \_\_fh[File\_Select - 1].FileName);

EXIT(0);

}

rewind(fp);

fclose(fp2);

delete tmp;

}

//==========================================

// 패킹파일에서 모든데이터를 읽어와 파일로 생성

//==========================================

else if (File\_Select == 0)

{

for (int i = 0; i < \_\_ph.FileCount; ++i)

{

char\* tmp = new char[\_\_fh[i].FileSize];

fseek(fp, \_\_fh[i].Offset, SEEK\_SET);

int res = fread(tmp, \_\_fh[i].FileSize, 1, fp);

if (res != true)

{

printf("[ 패킹파일 읽기 실패 ]");

EXIT(0);

}

FILE\* fp2;

errno\_t err2 = fopen\_s(&fp2, \_\_fh[i].FileName, "wb");

if (err2 != NULL)

{

printf("[ %s 파일생성 실패 ]", \_\_fh[i].FileName);

EXIT(0);

}

res = fwrite(tmp, \_\_fh[i].FileSize, 1, fp2);

if (res != true)

{

printf("[ %s 파일쓰기 실패 ]", \_\_fh[i].FileName);

EXIT(0);

}

rewind(fp);

fclose(fp2);

delete[] tmp;

}

}

delete[] \_\_fh;

printf("패킹 풀기완료\n\n");

}

else

{

printf("잘못된 입력");

EXIT(0);

}

return 0;

}

**<8.30과제>**

**더블버퍼링을 이용해서 텍스트게임 만들기 (움직이는 오브젝트)**

**Conolse.cpp**

#include "pch.h"

#include "Console.h"

HANDLE hConsole;

//-------------------------------------------------------------

// 이렇게 씁니다.

//

// #incude <stdio.h>

// #include <windows.h>

// #incude "Console.h"

//

// void main(void)

// {

// cs\_Initial();

//

// cs\_MoveCursor(0, 0); // 커서를 0, 0 위치로

// printf("abcde"); // 0, 0 위치에 글씨를 찍음

// cs\_MoveCursor(20, 10); // 커서를 20, 10 위치로

// printf("abcde"); // 0, 0 위치에 글씨를 찍음

//

// }

//-------------------------------------------------------------

//-------------------------------------------------------------

// 콘솔 제어를 위한 준비 작업.

//

//-------------------------------------------------------------

void cs\_Initial(void)

{

CONSOLE\_CURSOR\_INFO stConsoleCursor;

//-------------------------------------------------------------

// 화면의 커서를 안보이게끔 설정한다.

//-------------------------------------------------------------

stConsoleCursor.bVisible = FALSE;

stConsoleCursor.dwSize = 1; // 커서 크기.

// 이상하게도 0 이면 나온다. 1로하면 안나온다.

//-------------------------------------------------------------

// 콘솔화면 (스텐다드 아웃풋) 핸들을 구한다.

//-------------------------------------------------------------

hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleCursorInfo(hConsole, &stConsoleCursor);

}

//-------------------------------------------------------------

// 콘솔 화면의 커서를 X, Y 좌표로 이동시킨다.

//

//-------------------------------------------------------------

void cs\_MoveCursor(int iPosX, int iPosY)

{

COORD stCoord;

stCoord.X = iPosX;

stCoord.Y = iPosY;

//-------------------------------------------------------------

// 원하는 위치로 커서를 이동시킨다.

//-------------------------------------------------------------

SetConsoleCursorPosition(hConsole, stCoord);

}

//-------------------------------------------------------------

// 콘솔 화면을 조기화 한다.

//

//-------------------------------------------------------------

void cs\_ClearScreen(void)

{

int iCountX, iCountY;

DWORD dw;

//콘솔에다가 뭔가 출력하는것을 네트워크에서도 테스트용으로 종종써먹게된다.

//이때, 가끔 화면을 지우는것 자체가 너무느려서 화면이 10프레임도 안나오는경우가 종종 생긴다.

FillConsoleOutputCharacter(GetStdHandle(STD\_OUTPUT\_HANDLE), ' ', 100 \* 100, { 0, 0 }, &dw);

//-------------------------------------------------------------

// 화면 크기만큼 세로, 가로 이중 for 문을 사용하여

// 각각의 좌표마다 printf(" "); 공백을 전부 출력 해준다.

//-------------------------------------------------------------

for (iCountY = 0; iCountY < dfSCREEN\_HEIGHT; iCountY++)

{

for (iCountX = 0; iCountX < dfSCREEN\_WIDTH; iCountX++)

{

cs\_MoveCursor(iCountX, iCountY);

printf(" ");

}

}

}

**Conolse.h**

//////////////////////////////////////////////////////////////

// Windows 의 콘솔 화면에서 커서 제어.

//

//////////////////////////////////////////////////////////////

#ifndef \_\_CONSOLE\_\_

#define \_\_CONSOLE\_\_

#define dfSCREEN\_WIDTH 81 // 콘솔 가로 80칸 + NULL

#define dfSCREEN\_HEIGHT 24 // 콘솔 세로 24칸

//-------------------------------------------------------------

// 콘솔 제어를 위한 준비 작업.

//

//-------------------------------------------------------------

void cs\_Initial(void);

//-------------------------------------------------------------

// 콘솔 화면의 커서를 X, Y 좌표로 이동시킨다.

//

//-------------------------------------------------------------

void cs\_MoveCursor(int iPosX, int iPosY);

//-------------------------------------------------------------

// 콘솔 화면을 조기화 한다.

//

//-------------------------------------------------------------

void cs\_ClearScreen(void);

#endif

**Queue.cpp**

#include "pch.h"

#include "Queue.h"

void QueueInit(Queue\* pq)

{

pq->front = nullptr;

pq->rear = nullptr;

}

int QIsEmpty(Queue\* pq)

{

return (pq->front == nullptr) ||

(pq->rear == nullptr);

}

void Enqueue(Queue\* pq, Data data)

{

Node\* NewNode = new Node;

NewNode->data = data;

NewNode->next = nullptr;

if (QIsEmpty(pq))

{

pq->front = NewNode;

pq->rear = NewNode;

}

else

{

pq->rear->next = NewNode;

pq->rear = NewNode;

}

}

bool Dequeue(Queue\* pq, Data\* data)

{

if (QIsEmpty(pq))

{

//cout << "Under Flow" << endl;

return false;

}

Node\* rPos = pq->front;

\*data = rPos->data;

pq->front = pq->front->next;

delete rPos; rPos = nullptr;

return true;

}

Data QPeek(Queue\* pq)

{

return pq->front->data;

}

**Queue.h**

#pragma once

#ifndef \_\_QUEUE\_H\_\_

#define \_\_QUEUE\_H\_\_

typedef int Data;

typedef struct \_node

{

Data data = 0;

struct \_node\* next = nullptr;

}Node;

typedef struct \_lQueue

{

Node\* front = nullptr;

Node\* rear = nullptr;

}LQueue;

typedef LQueue Queue;

void QueueInit(Queue\* pq);

int QIsEmpty(Queue\* pq);

void Enqueue(Queue\* pq, Data data);

bool Dequeue(Queue\* pq, Data\* data);

Data QPeek(Queue\* pq);

#endif

**Assignment.cpp**

#include "pch.h"

#include "Console.h"

#include "Queue.h"

#define SCREEN\_HEIGHT 40

#define SCREEN\_WIDTH 81

#define UI\_HEIGHT 3 // UI 높이

#define RECORD\_HEIGHT 12 // 점수판 높이

#define WAIT\_FRAME\_1 100 // STAGE 글자출력 간격

#define WAIT\_FRAME\_2 300 // GAMEOVER 글자출력 간격

#define WAIT\_FRAME\_3 500 // GAMEOVER 출력 후 대기

#define LEVELUP\_EXP 1000

#define UPGRADE\_3 7 // 추가총알 나오는 레벨

#define STAGE\_TIME 1000 // 스테이지 전환간격

enum SCENE

{

TITLE\_SCENE = 10,

GAME\_SCENE = 20,

END\_SCENE = 30,

RECORD\_SCENE = 40,

};

struct ENEMY

{

bool valid = false;

char Id = 15;

int X, Y;

};

typedef struct ENEMY\_MAN

{

ENEMY Enemy[256];

int INDEX = 0;

int SPEED = 1;

}ENEMY\_MANAGEMENT;

struct PLAYER

{

int LIFE = 1;

int X, Y;

char Id = 127;

int LEVEL = 1;

int MAX\_EXP = 100;

int BOMB = 2;

int SCORE = 0;

char STAGE = 1;

int SIZE = 1;

};

struct BULLET

{

bool valid = false;

int X, Y;

char Id = ':';

};

struct PLUS\_BULLET

{

bool valid = false;

int X, Y;

char Id = 24;

};

typedef struct BULLET\_MAN

{

BULLET Bullet[64];

int INDEX = 0;

int SPEED = -2;

PLUS\_BULLET BulletP[64];

int INDEX\_P = 0;

int SPEED\_P = -2;

} BULLET\_MANAGEMENT;

enum INPUT\_COMMAND

{

MOVE\_RIGHT = 100, MOVE\_LEFT = 200,

MOVE\_UP = 300, MOVE\_DOWN = 400,

ACTION\_FIRE = 500,

ACTION\_BOMB = 600,

CHEAT\_LEVELUP = 700,

CHEAT\_NEXTSTAGE = 800,

};

char szScreenBuffer[SCREEN\_HEIGHT][SCREEN\_WIDTH];

\_\_int64 Frame = 0; // 프레임 측정

int WF\_COUNT = 0; // WaitFrameCount

int DIFFI\_COUNT; // 난이도 (적군 출현빈도) 조절

int QUIT; // 종료

PLAYER Player;

BULLET\_MAN BMT;

ENEMY\_MAN EMT;

SCENE Scene;

void INIT\_MAINGAME(void);

void TitleScene(void);

void Title\_Sprite(void);

void GameScene(void);

void Key\_Input(void);

void Apply\_Input(Queue\* q);

void Next\_Stage(void);

void CreateBullet(void);

void CreateBomB(void);

void Enemy\_Check(void);

void Bullet\_Check(void);

void Player\_Check(void);

void PLAYER\_LEVELUP(void);

void Sprite\_Draw(int X, int Y, char chSprite);

void Buffer\_Flip(void); // 버퍼내용을 화면으로 출력

void Buffer\_Clear(void); // 화면 버퍼를 지워주는 함수

void UI\_DRAW(void);

void EndScene(void);

void End\_Draw(void);

void RecordScene(void);

void Record\_Draw(void);

int main(void)

{

//게임시작전 초기화

INIT\_MAINGAME();

int a = 0;

while (QUIT == false)

{

if (Scene > RECORD\_SCENE)

Scene = GAME\_SCENE;

switch (Scene)

{

case TITLE\_SCENE:

TitleScene();

break;

case GAME\_SCENE:

GameScene();

break;

case END\_SCENE:

EndScene();

break;

case RECORD\_SCENE:

RecordScene();

break;

}

if ((GetAsyncKeyState(VK\_ESCAPE) & 0x8000) != 0)

{

Scene = TITLE\_SCENE;

Buffer\_Clear();

Buffer\_Flip();

break;

}

++Frame;

Sleep(20);

}

return 1;

}

/////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////

void INIT\_MAINGAME(void)

{

srand(unsigned(NULL));

cs\_Initial(); // 콘솔 초기화

Buffer\_Clear(); // 콘솔 클리어

Scene = TITLE\_SCENE; // 최초씬 설정

Player.X = (SCREEN\_WIDTH) / 2;

Player.Y = SCREEN\_HEIGHT - UI\_HEIGHT - 1; //플레이어 위치초기화

QUIT = 0;

}

void TitleScene()

{

Buffer\_Clear();

Title\_Sprite();

Buffer\_Flip();

if (\_kbhit() != false)

Scene = GAME\_SCENE;

}

void Title\_Sprite()

{

char TMP[SCREEN\_HEIGHT][SCREEN\_WIDTH] =

{

" ",

" ",

" ",

" ",

" ",

" ▣▣▣ ▣▣▣ ▣ ▣▣ ▣▣▣",

" ▣ ▣ ▣ ▣ ▣ ▣ ▣",

" ▣▣▣ ▣▣▣ ▣▣▣ ▣ ▣▣▣",

" ▣ ▣ ▣ ▣ ▣ ▣",

" ▣▣▣ ▣ ▣ ▣ ▣▣ ▣▣▣",

" ",

" □□□ □□□ □□□ □ □ □□□□",

" □ □ □ □ □ □ □ □",

" □ □ □□□ □□□ □ □ □ □□",

" □ □ □ □ □ □ □ □ □",

" □□□ □□□ □□□ □□□□ □□□□",

" ",

" ",

" ",

" ",

" FRESS < Any Key > TO START ",

" ",

" ",

" ◇ KEY INSTRUCTION ◇ CHEAT KEY",

" ",

" ",

" △ <L> : LEVEL UP",

" ◁ ▽ ▷ <N> : NEXT STAGE",

" ",

" ( MAX STAGE : 5 ) ",

" ( UPGRADE : 3,5,7 LEVEL ) ",

" ",

" KEY 'A' : Fire",

" SPACE : Bomb",

" ESC : Quit"

};

// 시작화면에서 깜빡임 효과

if (Frame % 30 < 15)

{

for (int i = 0; i < SCREEN\_HEIGHT; ++i)

{

if (i != 20)

strcpy\_s(szScreenBuffer[i], sizeof(szScreenBuffer[0]), TMP[i]);

}

}

else

for (int i = 0; i < SCREEN\_HEIGHT; ++i)

strcpy\_s(szScreenBuffer[i], sizeof(szScreenBuffer[0]), TMP[i]);

}

void GameScene()

{

// 스크린 버퍼를 지움

Buffer\_Clear();

// 플레이어 로직

Player\_Check();

// 적군로직

Enemy\_Check();

// 총알로직

Bullet\_Check();

// UI 출력

if (Scene == GAME\_SCENE)

UI\_DRAW();

// 일정시간 지나면 다음 스테이지

if (Frame % 1000 == 0)

Next\_Stage();

// 스크린 버퍼를 화면으로 출력

Buffer\_Flip();

// 프레임 맞추기용 대기 10 Frame

}

/////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////

// 키입력 감지

void Key\_Input()

{

Queue q;

QueueInit(&q);

// 입력이 감지되면 큐에 추가

if ((GetAsyncKeyState(VK\_RIGHT) & 0x8000) != false)

Enqueue(&q, MOVE\_RIGHT);

if ((GetAsyncKeyState(VK\_LEFT) & 0x8000) != false)

Enqueue(&q, MOVE\_LEFT);

if ((GetAsyncKeyState(VK\_UP) & 0x8000) != false)

Enqueue(&q, MOVE\_UP);

if ((GetAsyncKeyState(VK\_DOWN) & 0x8000) != false)

Enqueue(&q, MOVE\_DOWN);

// 액션

if ((GetAsyncKeyState(0x41) & 0x8000) != false)

Enqueue(&q, ACTION\_FIRE);

if ((GetAsyncKeyState(VK\_SPACE) & 0x8000) != false)

Enqueue(&q, ACTION\_BOMB);

//==================================================

// 치트키 입력

//==================================================

if ((GetAsyncKeyState(0x4C) & 0x8000) != false)

Enqueue(&q, CHEAT\_LEVELUP);

if ((GetAsyncKeyState(0x4E) & 0x8000) != false)

Enqueue(&q, CHEAT\_NEXTSTAGE);

Apply\_Input(&q);

}

//입력된 키 적용

void Apply\_Input(Queue\* q)

{

while (!QIsEmpty(q))

{

Data TData;

if (Dequeue(q, &TData) == true)

{

switch (TData)

{

case MOVE\_RIGHT:

if (Player.X < SCREEN\_WIDTH - 4) //마지막은 NULL이 들어가야 하기때문에.

++Player.X;

break;

case MOVE\_LEFT:

if (Player.X - Player.SIZE > 0)

--Player.X;

break;

case MOVE\_UP:

if (Player.Y > 0)

--Player.Y;

break;

case MOVE\_DOWN:

if (Player.Y < SCREEN\_HEIGHT - UI\_HEIGHT - 1)

++Player.Y;

break;

case ACTION\_FIRE:

CreateBullet();

break;

case ACTION\_BOMB:

CreateBomB();

break;

case CHEAT\_LEVELUP:

PLAYER\_LEVELUP();

break;

case CHEAT\_NEXTSTAGE:

Next\_Stage();

break;

}

}

}

}

void Next\_Stage(void)

{

WF\_COUNT = 0;

if (Player.STAGE < 5)

++Player.STAGE;

}

void CreateBullet(void)

{

// 기본 총알

BMT.Bullet[BMT.INDEX].valid = true;

BMT.Bullet[BMT.INDEX].X = Player.X;

BMT.Bullet[BMT.INDEX].Y = Player.Y;

if (BMT.INDEX >= \_countof(BMT.Bullet))

BMT.INDEX = 0;

else

++BMT.INDEX;

// 추가 총알 : 레벨이 7이상일 경우 추가총알 생성 //#define으로 빼기

if (Player.LEVEL >= UPGRADE\_3)

{

BMT.BulletP[BMT.INDEX\_P].valid = true;

BMT.BulletP[BMT.INDEX\_P].X = Player.X - 1;

BMT.BulletP[BMT.INDEX\_P].Y = Player.Y - 1;

if (BMT.INDEX\_P - 1 >= \_countof(BMT.BulletP))

BMT.INDEX\_P = 0;

else

++BMT.INDEX\_P;

BMT.BulletP[BMT.INDEX\_P].valid = true;

BMT.BulletP[BMT.INDEX\_P].X = Player.X + 1;

BMT.BulletP[BMT.INDEX\_P].Y = Player.Y - 1;

if (BMT.INDEX\_P >= \_countof(BMT.BulletP))

BMT.INDEX\_P = 0;

else

++BMT.INDEX\_P;

}

}

void CreateBomB(void)

{

for (int i = 0; i < \_countof(EMT.Enemy); ++i)

EMT.Enemy[i].valid = false;

}

/////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////

void Enemy\_Check(void)

{

//STAGE 출력

if (WF\_COUNT <= 100)

{

if (WF\_COUNT > WAIT\_FRAME\_1 - 90)

Sprite\_Draw(15, 20, 'S');

if (WF\_COUNT > WAIT\_FRAME\_1 - 85)

Sprite\_Draw(16, 20, 'T');

if (WF\_COUNT > WAIT\_FRAME\_1 - 80)

Sprite\_Draw(17, 20, 'A');

if (WF\_COUNT > WAIT\_FRAME\_1 - 75)

Sprite\_Draw(18, 20, 'G');

if (WF\_COUNT > WAIT\_FRAME\_1 - 70)

if (WF\_COUNT > WAIT\_FRAME\_1 - 70)

Sprite\_Draw(19, 20, 'E');

Sprite\_Draw(20, 20, 32);

if (WF\_COUNT > WAIT\_FRAME\_1 - 65)

Sprite\_Draw(21, 20, Player.STAGE + 48); //49는 아스키코드 '1'

// 게임시작후 WAIT\_FRAME\_1만큼 기다렸다가 적군 출현

if (++WF\_COUNT < WAIT\_FRAME\_1)

return;

for (int i = 0; i < \_countof(EMT.Enemy); ++i)

EMT.Enemy[i].valid = false;

}

//유효한 적군만 Sprite

for (int i = 0; i < \_countof(EMT.Enemy); ++i)

{

// 적군이 유효하다면

if (EMT.Enemy[i].valid == true)

{

//==================================================

// Enemy와 기본 총알 충돌 ==> 적군 파괴

//==================================================

for (int j = 0; j < \_countof(BMT.Bullet); ++j)

{

//총알이 유효하다면

if (BMT.Bullet[j].valid == true)

{

if ((EMT.Enemy[i].X == BMT.Bullet[j].X && EMT.Enemy[i].Y == BMT.Bullet[j].Y - 1)

|| EMT.Enemy[i].X == BMT.Bullet[j].X && EMT.Enemy[i].Y == BMT.Bullet[j].Y)

{

EMT.Enemy[i].valid = false;

BMT.Bullet[i].valid = false;

Sprite\_Draw(BMT.Bullet[i].X, BMT.Bullet[i].Y, 32); // 폭발 효과 Sprite

if (Player.LIFE > 0)

Player.SCORE += 10;

}

}

}

//==================================================

// Enemy와 추가 총알 충돌 ==> 적군 파괴

//==================================================

for (int j = 0; j < \_countof(BMT.BulletP); ++j)

{

//총알이 유효하다면

if (BMT.Bullet[j].valid == true)

{

if ((EMT.Enemy[i].X == BMT.BulletP[j].X && EMT.Enemy[i].Y == BMT.BulletP[j].Y - 1)

|| EMT.Enemy[i].X == BMT.BulletP[j].X && EMT.Enemy[i].Y == BMT.BulletP[j].Y)

{

EMT.Enemy[i].valid = false;

BMT.BulletP[i].valid = false;

Sprite\_Draw(BMT.BulletP[i].X, BMT.BulletP[i].Y, 32); // 폭발 효과 Sprite

if (Player.LIFE > 0)

Player.SCORE += 10;

}

}

}

//==================================================

// 적군이 화면밖으로 나간경우 : 삭제처리

//==================================================

if (EMT.Enemy[i].Y + EMT.SPEED >= SCREEN\_HEIGHT - UI\_HEIGHT)

EMT.Enemy[i].valid = false;

//==================================================

// Enemy와 플레이어가 충돌 : 플레이어 라이프감소

//==================================================

else if ((Player.X == EMT.Enemy[i].X && EMT.Enemy[i].Y + EMT.SPEED == Player.Y)

|| (Player.X == EMT.Enemy[i].X && EMT.Enemy[i].Y == Player.Y))

//

{

if (Player.LIFE > 0)

{

Sprite\_Draw(Player.X, Player.Y, 38);

--Player.LIFE;

}

}

//==================================================

// (적군) 아무이벤트가 없다면 Sprite

//==================================================

else

{

EMT.Enemy[i].Y += EMT.SPEED;

Sprite\_Draw(EMT.Enemy[i].X, EMT.Enemy[i].Y, EMT.Enemy[i].Id);

}

//==================================================

// 플레이어가 업그레이드 상태(2) 일때 적군과 충돌

//==================================================

if (Player.LEVEL >= 5)

{

if ((Player.X + 1 == EMT.Enemy[i].X && EMT.Enemy[i].Y + EMT.SPEED == Player.Y)

|| (Player.X + 1 == EMT.Enemy[i].X && EMT.Enemy[i].Y == Player.Y))

{

--Player.LIFE;

EMT.Enemy[i].valid = false;

}

if ((Player.X - 1 == EMT.Enemy[i].X && EMT.Enemy[i].Y + EMT.SPEED == Player.Y)

|| (Player.X - 1 == EMT.Enemy[i].X && EMT.Enemy[i].Y == Player.Y))

{

--Player.LIFE;

EMT.Enemy[i].valid = false;

}

}

//==================================================

// 플레이어가 업그레이드 상태(3) 일때 적군과 충돌

//==================================================

if (Player.LEVEL >= UPGRADE\_3)

{

if ((Player.X + 2 == EMT.Enemy[i].X && EMT.Enemy[i].Y + EMT.SPEED == Player.Y)

|| (Player.X + 1 == EMT.Enemy[i].X && EMT.Enemy[i].Y == Player.Y))

{

--Player.LIFE;

EMT.Enemy[i].valid = false;

}

if ((Player.X - 2 == EMT.Enemy[i].X && EMT.Enemy[i].Y + EMT.SPEED == Player.Y)

|| (Player.X - 1 == EMT.Enemy[i].X && EMT.Enemy[i].Y == Player.Y))

{

--Player.LIFE;

EMT.Enemy[i].valid = false;

}

}

}

}

//=========================================

// 스테이지에 따른 적군 조절 : 난이도 조절

//=========================================

DIFFI\_COUNT = (Player.STAGE);

while (DIFFI\_COUNT--)

{

EMT.Enemy[EMT.INDEX].valid = true;

EMT.Enemy[EMT.INDEX].Y = 0;

EMT.Enemy[EMT.INDEX].X = rand() % (SCREEN\_WIDTH - 2);

++EMT.INDEX;

if (EMT.INDEX >= \_countof(EMT.Enemy))

EMT.INDEX = 0;

}

}

//==================================================

// 총알 Sprite

//==================================================

void Bullet\_Check(void)

{

// 기본 총알 출력

for (int i = 0; i < \_countof(BMT.Bullet); ++i)

{

// 만약 유효한 총알이라면

if (BMT.Bullet[i].valid == true)

{

// 총알이 스크린밖으로 나갈경우 총알 무효화

if (BMT.Bullet[i].Y - BMT.SPEED < 0)

BMT.Bullet[i].valid = false;

// 그게아니라면 총알에 속도를 더한다.

else

{

BMT.Bullet[i].Y += BMT.SPEED;

Sprite\_Draw(BMT.Bullet[i].X, BMT.Bullet[i].Y, BMT.Bullet[i].Id);

}

}

}

// 추가 총알

for (int i = 0; i < \_countof(BMT.BulletP); ++i)

{

if (BMT.BulletP[i].valid == true)

{

if (BMT.BulletP[i].Y - BMT.SPEED\_P <= 0)

BMT.BulletP[i].valid = false;

else

{

BMT.BulletP[i].Y += BMT.SPEED\_P;

Sprite\_Draw(BMT.BulletP[i].X, BMT.BulletP[i].Y, BMT.BulletP[i].Id);

}

}

}

}

void Player\_Check(void)

{

//==================================================

// 플레이어가 사망시 씬 전환

//==================================================

if (Player.LIFE <= 0)

{

Player.X = 0, Player.Y = 0;

Scene = END\_SCENE;

return;

}

// 사용자 키입력

Key\_Input();

// 플레이어 스프라이트

Sprite\_Draw(Player.X, Player.Y, Player.Id);

// 플레이어 레벨업

if (Player.SCORE > Player.MAX\_EXP)

PLAYER\_LEVELUP();

}

//==================================================

// 플레이어 레벨 업

//==================================================

void PLAYER\_LEVELUP()

{

//레벨업했다면 최대 경험치 증가

++Player.LEVEL;

Player.MAX\_EXP = (Player.LEVEL + Player.LEVEL) \* LEVELUP\_EXP;

// 레벨이 3이상일경우, 비행기사이즈 증가 + 총알밀집도 증가

if (Player.LEVEL >= 3)

{

Player.SIZE = 1;

if (BMT.SPEED == -2)

++BMT.SPEED;

}

// 레벨이 5 이상일 경우 기본총알 중 50% 업그레이드

if (Player.LEVEL >= 5)

{

Player.SIZE = 2;

for (int i = 0; i < \_countof(BMT.Bullet); ++i)

{

if (i % 2 == 0)

BMT.Bullet[i].Id = 'o';

}

}

// 레벨이 7이상일 경우 CreateBullet함수에서 추가총알 생성

}

///////////////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////////////

void Sprite\_Draw(int X, int Y, char chSprite)

{

if (Y - 1 > SCREEN\_HEIGHT || X - 1 > SCREEN\_WIDTH)

return;

szScreenBuffer[Y][X] = chSprite;

//플레이어 업그레이드

if (chSprite == Player.Id && Player.LEVEL > 7)

{

szScreenBuffer[Y][X + 2] = 19;

szScreenBuffer[Y][X + 1] = ']';

szScreenBuffer[Y][X - 1] = '[';

szScreenBuffer[Y][X - 2] = 19;

}

else if (chSprite == Player.Id && Player.LEVEL > 5)

{

szScreenBuffer[Y][X + 2] = '\*';

szScreenBuffer[Y][X + 1] = ']';

szScreenBuffer[Y][X - 1] = '[';

szScreenBuffer[Y][X - 2] = '\*';

}

else if (chSprite == Player.Id && Player.LEVEL > 3)

{

szScreenBuffer[Y][X + 1] = ']';

szScreenBuffer[Y][X - 1] = '[';

}

}

void Buffer\_Flip(void)

{

// 레코드씬이 아닐경우 그냥 출력

int i;

if (Scene != RECORD\_SCENE)

{

for (i = 0; i < SCREEN\_HEIGHT - UI\_HEIGHT; ++i)

{

cs\_MoveCursor(0, i);

printf(szScreenBuffer[i]);

}

}

// 레코드씬일경우 점수판을 출력

else

{

for (i = 0; i < SCREEN\_HEIGHT; ++i)

{

cs\_MoveCursor(0, i);

printf(szScreenBuffer[i]);

}

cs\_MoveCursor(0, RECORD\_HEIGHT);

//파일로 이전 최고점수기록 불러오기 (못함)

printf("\n\n\t\tDATE\t\tID\tSTAGE\t\tSCORE");

printf("\n\n\n\t\t%s\t %c\t %d\t\t %d", \_\_DATE\_\_, Player.Id

, Player.STAGE, Player.SCORE);

}

}

void Buffer\_Clear(void)

{

memset(szScreenBuffer, 32, sizeof(szScreenBuffer));

for (int i = 0; i < SCREEN\_HEIGHT; ++i)

szScreenBuffer[i][SCREEN\_WIDTH - 1] = '\0';

}

void UI\_DRAW()

{

int i = SCREEN\_HEIGHT - UI\_HEIGHT;

// 첫째줄

cs\_MoveCursor(0, i);

printf("┌");

for (int i = 0; i < SCREEN\_WIDTH - 3; ++i)

printf("─");

printf("┐");

// 둘째줄 : 점수판

cs\_MoveCursor(0, ++i);

printf("│ LIFE ");

for (int i = 1; i < Player.LIFE; ++i)

printf("%c", 127);

printf(" BOMB ");

for (int i = 0; i < Player.BOMB; ++i)

printf("◎");

printf(" LEVEL :");

printf("%d", Player.LEVEL);

printf(" STAGE %d", Player.STAGE);

printf("\t\tSCORE %d", Player.SCORE);

cs\_MoveCursor(SCREEN\_WIDTH - 2, i);

printf("│");

// 셋째줄

cs\_MoveCursor(0, ++i);

printf("└");

for (int i = 0; i < SCREEN\_WIDTH - 3; ++i)

printf("─");

printf("┘");

}

void EndScene(void)

{

Buffer\_Clear();

Bullet\_Check();

End\_Draw();

Buffer\_Flip();

}

void End\_Draw()

{

if (WF\_COUNT < WAIT\_FRAME\_3)

{

++WF\_COUNT;

Enemy\_Check();

int X = (SCREEN\_WIDTH / 2) - 7;

if (WF\_COUNT > WAIT\_FRAME\_2 - 150) //101 210

Sprite\_Draw(X + 1, 20, 'G');

if (WF\_COUNT > WAIT\_FRAME\_2 - 140)

Sprite\_Draw(X + 2, 20, 'A');

if (WF\_COUNT > WAIT\_FRAME\_2 - 130)

Sprite\_Draw(X + 3, 20, 'M');

if (WF\_COUNT > WAIT\_FRAME\_2 - 120)

Sprite\_Draw(X + 4, 20, 'E');

Sprite\_Draw(X + 6, 20, ' ');

if (WF\_COUNT > WAIT\_FRAME\_2 - 90)

Sprite\_Draw(X + 7, 20, 'O');

if (WF\_COUNT > WAIT\_FRAME\_2 - 80)

Sprite\_Draw(X + 8, 20, 'V');

if (WF\_COUNT > WAIT\_FRAME\_2 - 70)

Sprite\_Draw(X + 9, 20, 'E');

if (WF\_COUNT > WAIT\_FRAME\_2 - 50)

Sprite\_Draw(X + 10, 20, 'R');

}

else if (WF\_COUNT >= WAIT\_FRAME\_3)

Scene = RECORD\_SCENE;

}

void RecordScene(void)

{

Buffer\_Clear();

Record\_Draw();

Buffer\_Flip();

}

void Record\_Draw(void)

{

char TMP[RECORD\_HEIGHT][SCREEN\_WIDTH] =

{

" ",

" ",

" ",

" ",

" ",

" ▣▣▣ ▣▣▣ ▣▣▣▣ ▣▣▣ ▣▣▣ ",

" ▣ ▣ ▣ ▣ ▣ ▣ ▣",

" ▣▣▣ ▣ ▣ ▣ ▣▣▣ ▣▣▣",

" ▣ ▣ ▣ ▣ ▣ ▣ ▣",

" ▣▣▣ ▣▣▣ ▣▣▣▣ ▣ ▣ ▣▣▣",

" ",

" =============================================== ",

};

for (int i = 0; i < RECORD\_HEIGHT; ++i)

{

strcpy\_s(szScreenBuffer[i], sizeof(szScreenBuffer[0]), TMP[i]);

}

}**Pch.h**

// pch.h: 미리 컴파일된 헤더 파일입니다.

// 아래 나열된 파일은 한 번만 컴파일되었으며, 향후 빌드에 대한 빌드 성능을 향상합니다.

// 코드 컴파일 및 여러 코드 검색 기능을 포함하여 IntelliSense 성능에도 영향을 미칩니다.

// 그러나 여기에 나열된 파일은 빌드 간 업데이트되는 경우 모두 다시 컴파일됩니다.

// 여기에 자주 업데이트할 파일을 추가하지 마세요. 그러면 성능이 저하됩니다.

#ifndef PCH\_H

#define PCH\_H

#include <iostream>

#include <time.h>

#include <conio.h>

#include <windows.h>

#endif //PCH\_H