게임 자료구조와 알고리즘

-CHAPTER3-

SOULSEEK

목차

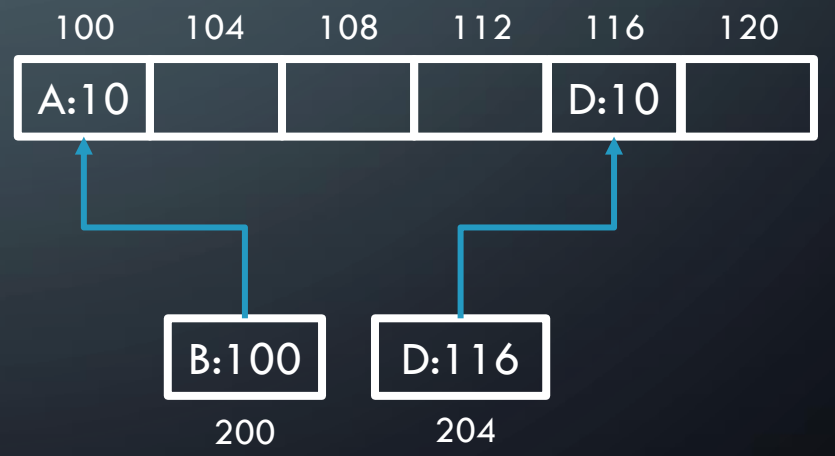
1. POINTER(포인터)
2. STRUCTURE(구조체)
3. REFERENCES
4. #DEFINE & TYPEDEF
5. ENUM(열거체)

POINTER(포인터)

1. POINTER(포인터)

- 다른 **변수의 위치(주소)**를 가리키는 변수.
- 포인터 변수의 자료형은 **가리키는 변수의 자료형과 동일해야** 한다.
  - 포인터 변수를 선언할 때 : **자료형* 변수명 = 자료형 *변수명**
  - 주소를 할당 받을 때 : 포인터 변수 = **&주소전달 할 변수**
  - 사용하는 곳에서 사용할 때 : cout << ***포인터 변수** << endl
- 포인터 변수의 **크기는 4byte**이다.
- 포인터 변수는 **NULL을 이용해 항상 초기화**해야 한다.

```
Int   A, D = 10;
Int*  B = NULL;
Int   *C = NULL;

B = A;
C = D;

cout << " A의 값은 : " << *B << endl
cout << " D의 값은 : " << *C << endl
```



1. POINTER(포인터)

포인터의 초기화와 사용

```
#include<iostream>
using namespace std;

void main()
{
	int nNo;
	int* pNo = NULL;

	nNo = 10;
	pNo = &nNo;

	cout << "nNo : " << nNo << endl;
	cout << "pNo : " << pNo << endl;
	cout << "nNo : " << *pNo << endl;

	*pNo = 100;

	cout << "nNo : " << *pNo << endl;
}
```

1. POINTER(포인터)

다음 구문을 풀이해 보세요

```
void main()
{
	int nNo = 10, nNo2 = 20;
	int* pNo = NULL;

	pNo = &nNo2;

	*pNo += 100;
	*pNo -= *pNo - nNo++;

	pNo = &nNo2;
	nNo *= *pNo;

	(*pNo)--;

	cout << "nNo : " << nNo << "nNo2 : " << nNo2 << "*pNo : " << *pNo << endl;
}
```

1. POINTER(포인터)

포인터 자료형의 크기

```
void main()
{
	int nNum = 10;
	char cCh = 'a';

	int* pNum = NULL;
	char* pCh = NULL;

	pNum = &nNum;
	pCh = &cCh;

	cout << "char 자료형 크기 : " << sizeof(cCh) << endl;
	cout << "int 자료형 크기 : " << sizeof(nNum) << endl;
	cout << "char* 자료형 크기 : " << sizeof(pCh) << endl;
	cout << "int* 자료형 크기 : " << sizeof(pNum) << endl;
}
```

1. POINTER(포인터)

포인터와 배열

- 포인터 변수에 배열의 시작주소를 넣어서 배열 전체를 컨트롤 할 수 있다.
  - 배열의 이름은 첫번째 원소의 주소다.
  - 배열의 첫번째 원소를 가리키고 있기 때문에 포인터의 덧셈, 뺄셈으로 배열의 원소를 가리킬 수 있다.

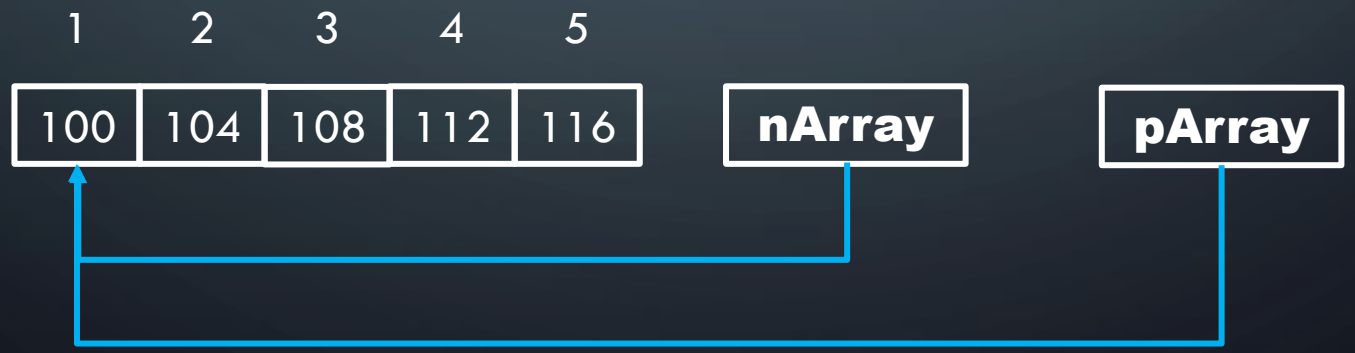
- long* p[20] = long타입의 주소를 가지는 원소 20개
  - [] 연산자가 우선순위이기 때문에 long* 타입의 포인터 변수 20개가 된다.

- long (*p)[20] = long타입의 원소20개를 가지는 배열
  - ()우선 순위를 가지가 때문에 long[20] 배열에 대한 포인터가 되는 것이다.

- 포인터는 다른 원소를 가리키게 변경할 수 있지만, 배열의 이름은 항상 첫번째 원소의 주소만을 의미하는 상수이다.

1. POINTER(포인터)

```
void main()
{
	int nArray[5];
	int* pArray = nArray;
	//int* pArray = &nArray[0];

	for (int i = 0; i < 5; ++i)
	{
		*(pArray + i) = i;

		cout << "Array[" << i << "] : " << nArray[i] << endl;
	}
}
```



1. POINTER(포인터)

포인터의 포인터

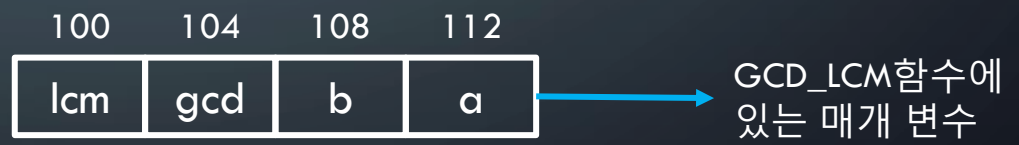
```
void main()
{
	int Num = 10;
	int *pNum = &Num;
	int* *ppNum = &pNum;
	int** *pppNum = &ppNum;

	cout << "Num : " << Num << ", &Num : " << &Num << endl;

	cout << "*pNum : " << *pNum << ", pNum : " << pNum;
	cout << ", &pNum : " << &pNum << endl;

	cout << "**ppNum : " << **ppNum << ", *ppNum : " << *ppNum;
	cout <<", ppNum : " << ppNum << "&ppNum : " << &ppNum << endl;

	cout << "***pppNum : " << ***pppNum << ", **pppNum : " << **pppNum;
	cout << ", *pppNum : " << *pppNum << ", pppNum : " << pppNum;
	cout << ", &pppNum : " << &pppNum << endl;
}
```

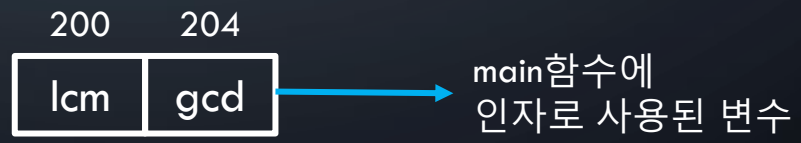
1. POINTER(포인터)

함수 인자로 포인터의 전달

```
void GCD_LCM(int a, int b, int gcd, int lcm)
{
	int z;
	int x = a;
	int y = b;

	while (true)
	{
		z = x % y;

		if (0 == z)
			break;

		x = y;
		y = z;
	}

	gcd = y;
	lcm = a * b / gcd;
}
```

```
int main()
{
	int gcd = 0;
	int lcm = 0;

	GCD_LCM(28, 35, gcd, lcm);

	cout << "GCD = " << gcd << endl;
	cout << "LCM = " << lcm << endl;

	return 0;
}
```





1. POINTER(포인터)

```
void GCD_LCM(int a, int b, int* pgcd, int* plcm)
{
	int z;
	int x = a;
	int y = b;

	while (true)
	{
		z = x % y;

		if (0 == z)
			break;

		x = y;
		y = z;
	}

	*pgcd = y;
	*plcm = a * b / *pgcd;
}
```

```
int main()
{
	int gcd = 0;
	int lcm = 0;

	GCD_LCM(28, 35, &gcd, &lcm);

	cout << "GCD = " << gcd << endl;
	cout << "LCM = " << lcm << endl;

	return 0;
}
```

학습과제

- 문자열을 입력 받고 입력 받은 문자열을 대문자로 바꾸는 함수를 만들어 보자.
- 두 수를 입력 받아 큰 수를 출력해 보자.
- 하나의 수를 입력 받아서 1부터 받은 수까지의 합을 구해보자.
- 임의의 수를 8개를 int형 배열에 입력 받은 후 오름차순, 내림차순 정렬한 뒤에 출력해보자.
- 미로 찾기나 별똥별 피하기 코드에서 포인터로 바꿔서 적용할 수 있는 부분을 찾아서 적용시켜보자.

조건

1. main 함수에는 cout이나 cin같은 입출력 구분만 존재한다.
2. 함수를 이용해서 전달받고 return 없이 연산만 해야 한다.

STRUCTURE(구조체)

2. STRUCTURE

- 하나의 오브젝트의 특징을 나타내는 여러 자료형의 변수를 모아서 **그룹**처럼 만드는데 사용.
- **사용자 정의 타입**(User Defined Type)이다.
- 실존 정보를 데이터화 하는 **추상화** 작업이다.
- 구조체가 가지고 있는 각 변수를 **멤버변수**라고 부른다.
- 구조체는 **배열의 초기화 방법**을 그대로 활용할 수 있다.

```
struct 구조체 이름
{
    자료형 멤버변수
    자료형 멤버변수
    자료형 멤버변수
};
```

```
struct Player
{
    char Name[12];
    int Level;
    int Hp;
};
```

2. STRUCTURE

기본 정의

```
struct Character
{
	char NickName[12];
	int Level;
	int Hp;
};

void main()
{
	struct Character player = {“SoulSeek”, 99, 10000};

	cout << “닉네임 : ” << player.NickName << “, 레벨 : ”
		<< player.Level << “, 체력 : ” << player.Hp << endl;
}
```

2. STRUCTURE

구조체의 배열

```
struct Character
{
	char NickName[12];
	int Level;
	int Hp;
};

void ShowPlayerInfo(Character p)
{
	cout << “★☆★☆★☆★☆★☆★☆” << endl;
	cout << “닉네임 : ” << p.NickName << endl;
	cout << “레벨 : ” << p.Level << endl;
	cout << “체력 : ” << p.Hp << endl;
	cout << “★☆★☆★☆★☆★☆★☆” << endl;
}
```

```
void main()
{
	Character P_List[3];

	for(int i = 0; i < 3; i++)
	{
		cout << “★☆★☆” << i << “번째 ★☆★☆\n”;
		cout << “닉네임 입력 : ”;
		cin >> P_List[i].NickName;
		cout << “레벨 입력 : ”;
		cin >> P_List[i].Level;
		cout << “체력 입력 : ”;
		cin >> P_List[i].Hp;
	}

	for(int i = 0; i < 3; i++)
	{
		ShowPlayerInfo(P_List[i]);
	}
}
```

2. STRUCTURE

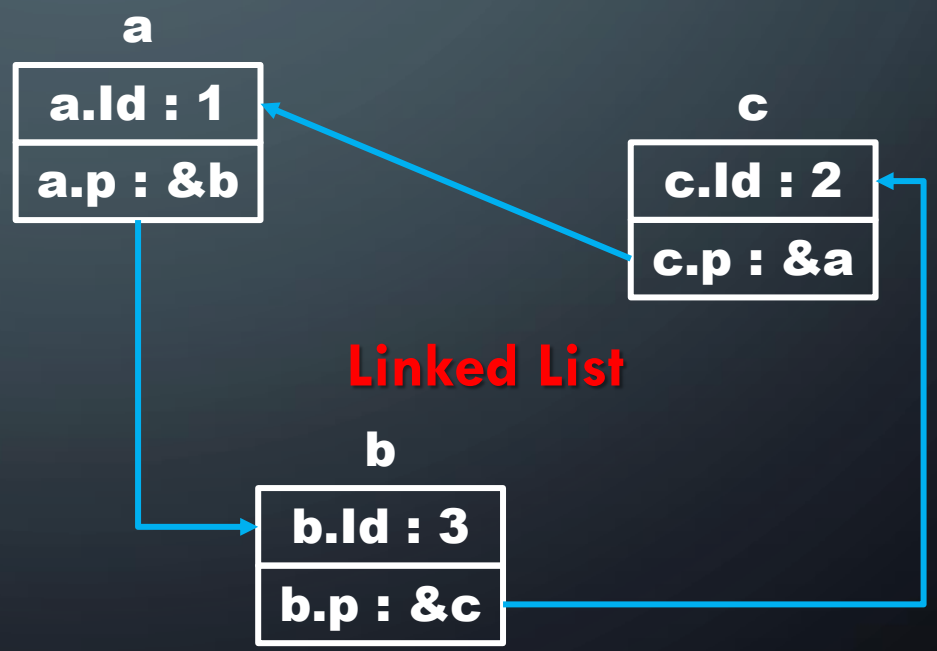
구조체를 가리키는 포인터

```
struct Rectangle
{
	int x, y;
	int width, height;
};

void main()
{
	Rectangle rc = {100, 100, 50, 50};

	Rectangle* p = &rc;

	p->y = 250; //(*p).x와 같은 의미 -  이미 배열과 포인터의 관계에서 배웠다.

	cout << “rc = (” << rc.x << “, ” << rc.y << “, ”
	cout << rc.width << “, ” << rc.height << “)” << endl;
}
```

2. STRUCTURE

포인터를 포함하는 구조체

```
struct Dizzy
{
	int id;
	Dizzy* p;
};

void main()
{
	Dizzy a, b, c;

	a.id = 1;
	a.p = &b;
	b.id = 2;
	b.p = &c;
	c.id = 3;
	c.p = &a;

	cout << “a.id : ” << a.id << endl;
	cout << “b.id : ” << a.p->id << endl;
	cout << “c.id : ” << a.p->p->id << endl;
	cout << “a.id : ” << a.p->p->p->id << endl;
}
```



2. STRUCTURE

함수의 인자로 전달하는 구조체

```
struct Point
{
	int x, y;
};

double Distance(Point* p1, Point* p2);

void main()
{
	Point a = {0, 0};
	Point b = {3, 4};

	double dist_a_b = Distance(&a, &b);
}
```

```
double Distance(Point* p1, Point* p2)
{
	cout << “p1 = ” << p1->x << “, ”
		<< p1->y << endl;

	cout << “p2 = ” << p2->x << “, ”
		<< p2->y << endl;

	return 0;
}
```

- 인자 값도 자료형을 가지고 메모리를 차지하고 있기때문에 구조체 전체의 멤버변수를 사용하는게 아니라면 메모리상의 주소만 넘기고 사용하자.

2. STRUCTURE

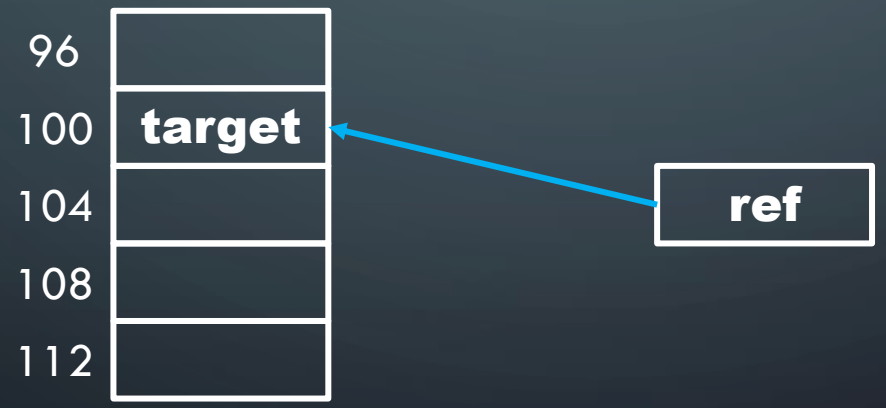
학습과제

- 위의 예시에 나왔던 캐릭터 정보를 입력하는 구조체를 이용해서 정보를 입력 받고 오름차순 내림차순 정렬을 출력하는 프로그램을 작성하라.
- 이전에 만들었던 미로 찾기나 별똥별 피하기 게임에서 하나로 합칠 수 있는 그룹화 가능한 변수들을 구조체로 변경해보자.

REFERENCES

3. REFERENCES

- 특정 변수의 이름에 **별명**을 부여한다.
- 선언할 때 반드시 **초기값**이 있어야한다.
- 자료형& 변수명 = 변수 – int& ref = target;
- 함수에서 매개변수를 레퍼런스 변수로 인자 값을 받아서 **원본의 정보처리로 진행**할 수 있다.
  - Referendce가 원래의 변수를 가리키고 있기 때문이다.



3. REFERENCES

```
int main()
{
	int target = 20;

	int& ref = target;

	cout << "ref= " << ref << "\n";
	cout << "target = " << target << "\n";
	cout << "&ref= " << &ref << "\n";
	cout << "&target= " << &target << "\n";

	ref = 100;

	cout << "ref= " << ref << "\n";
	cout << "target = " << target << "\n";

	return 0;
}
```

3. REFERENCES

함수의 인자 값 전달 시 극대화!

```
void GCD_LCM(int a, int b, int& gcd, int& lcm)
{
	int z;
	int x = a;
	int y = b;

	while (true)
	{
		z = x % y;

		if (z == 0)
		break;

		x = y;
		y = z;

		gcd = y;
		lcm = a * b / gcd;
	}
}
```

```
int main()
{
	int gcd = 0;
	int lcm = 0;
	GCD_LCM(28, 35, gcd, lcm);

	cout << "GCD = " << gcd << "\n";
	cout << "LCM = " << lcm << "\n";

	return 0;
}
```

- 함수 인자로 전달 될 경우에는 포인터이냐 레퍼런스냐 고민 될 경우가 많다. **레퍼런스는 반드시 초기화**를 하여 대상을 가지고 있지만 **포인터는 항상 대상을 가지고 있지 않을 수도 있고 NULL또는 쓰레기 값을 검증** 할 필요가 있다 그렇기 때문에 **인자전달을 할 경우 레퍼런스를 추천한다.**

3. REFERENCES

학습과제

- 두 개의 정수를 입력 받아 큰 수를 출력해보자.
- 하나의 수를 입력 받아 1 ~ 100사이의 해당 수의 배수 합을 구해보자.
  - ex) 5를 입력하면 5의 배수가 모두 출력된다.
- 하나의 숫자를 입력 받아서 거꾸로 출력해보자.

*조건 – 포인터와 똑같이 main함수에는 출력 부분만 존재 하고 return 은 사용하지 말자.

#DEFINE & TYPEDEF

4. #DEFINE & TYPEDEF

Typedef

- 특정 자료형에 별명을 붙여준다.
- 가독성 향상에 도움을 준다.
- 원형이 무엇인지 알고 있는 것이 코딩에 도움이 된다.(각 OS의 API)

```
#include <iostream>
using namespace std;
typedef int INT;
typedef int* PINT;

void main()
{
	INT a = 10;
	cout << "a = " << a << endl;
	PINT pA = &a;
	cout << "a 주소 : " << pA << "a 값 : " << *pA << endl;
}
```

4. #DEFINE & TYPEDEF

#define

- 형식 정의를 위한 전처리문
- 상수 값을 지정하기 위한 매크로, 구문 중간에 값 변경이 불가하다
- 여러 곳에 사용되며 다른 상수들과 구분될 필요성이 있을 때 사용
- 쓸데없는 #define 남발은 메모리 낭비를 가져온다
  - 단지 한 코드블록단위에서만 사용하기 위한 매크로는 만들지 않는 것이 좋다. 그럴 경우에는 C++이라면 const, C#라면 readonly를 사용하자.
  - 프로젝트 단위 혹은 솔루션 단위에서 default로 정의해서 사용하는 경우 매크로를 활용하면 정말 유용하다. 이럴 때는 클래스에 const로 모두 담아서 쓸 경우보다 리소스 소비가 줄어들 것이다.

4. #DEFINE & TYPEDEF

```
#define PI 3.14
//const float PI = 3.14;

void main()
{
	double tmp;

	cout << "PI = " << PI << endl;
	//cout << "PI = " << PI << endl;

	tmp = PI + 1;
	//tmp = 3.14 + 1;

	cout << "tmp = " << tmp << endl;
}
#define DebugLog(x) { cout << "들어온 값은 = " << x << endl; }

void main()
{
	int x = 5;

	DebugLog(x);
}
```

ENUM(열거체)

5. ENUM(열거체)

- 하나의 타입이며 정수 타입의 형태이다.
- 순차적인 나열을 하지 않을 수 있으며 시작 지점의 원소가 0이 아니어도 된다.
- 변수명을 선언할 때, 대문자를 주로 사용한다(필수는 아니지만 암묵적인 룰이다.)
- 순차적일 수도 아닐 수도 있는 형태이지만 마지막지점(MAX의 지점)은 정해두자.

```
enum JOB
{
	JOB_WARRIOR,
	JOB_SORCERER,
	JOB_ARCHER,
};
```

```
enum GAME_STATE
{
	GAME_START = 10,
	GAME_PLAY,
	GAME_STOP,

	GAME_END = 99,
};
```

5. ENUM(열거체)

```
if (bGameStart)
{

}
else
{

}

if (bGamePlay)
{

}
else
{

}
```

```
if (bGameStop)
{

}
else
{

}

if (bGameEnd)
{

}
else
{

}
```

```
switch (gameState)
{
    case GAME_START:
    break;
    case GAME_PLAY:
    break;
    case GAME_STOP:
    break;
    case GAME_END:
    break;
}
```

5. ENUM(열거체)

참고코드를 확인하여 사용예를 확인하자. 정말 유용하게 활용할 수 있다.

5. ENUM(열거체)

학습과제

- 미로 찾기나 별똥별 피하기에서 여러 상태를 설정할 때 reference, define, typedef, enum으로 사용 할 수 있는 것들을 변경해 보자.