ㅋ

|  |
| --- |
| **일일 업무 사항 정리** |

|  |  |
| --- | --- |
| **작성자** | 제품팀 이민성 인턴 |
| **업무 일시** | 20230109~20230113 |

|  |
| --- |
| **세부 사항** |
| **1. 업무 내역 요약 정리**   |  |  | | --- | --- | | **목표 내역** | **Done & Plan** | |  | **인공지능 vs 네트워크 프로그래밍**  **🡪 컴퓨터와 플레이하게끔 만들기로 결정하였습니다.**  **다차원 배열과 시간 함수에 대한 공부가 부족해 추가적으로 공부하였습니다.**  **틱택토 프로젝트 계획서 PPT는 메일로 첨부하였습니다.**  **부족한 점이 있으면 추가적인 공부를 하며 프로그램을 개발 할 계획입니다.**  **1. 다차원 배열**  **1-1. 2차원 배열**  **1-2. 3차원 배열**  **2. 시간 함수**  **3. 헤더파일**  **4. 미니맥스 알고리즘**  **5. 난수** | |

**2. 내용 세부 (업무 세부 내역 정리 및 기타 사항 정리)**

1. 다차원 배열

**1-1. 2차원 배열**

- 2차원 배열

다음 두 2차원 배열의 선언을 보자.

int arr1[3][4]; // 세로가 3, 가로가 4인 int형 2차원 배열

int arr2[2][6]; // 세로가 2, 가로가 6인 int형 2차원 배열

위의 두 선언에서 보이듯이 2차원 배열의 선언방식은 1차원 배열의 선언방식과 매우 유사하다. 다만 그 특성상 세로와 가로의 길이를 각각 명시하는 형태를 띨 뿐이다. 따라서 위와 같이 배열이 선언되면 다음의 형태로 배열이 생성된다.

텍스트, 전자기기이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명

위 그림에서 배열요소 안에 삽입된 두 개의 숫자는 각각의 요소에 접근할 때 사용하게 되는 인덱스 값이다. 정리하면 배열의 이름이 arr이고 배열요소의 자료형이 TYPE이라 할 때, 2차원 배열의 선언형태는 다음과 같다.

TYPE arr[세로길이][가로길이];

sizeof 연산자의 펴연산자로 배열의 이름이 오면, 배열의 크기가 바이트 단위로 계산되어서 반환된다. 이를 이용해서 2차원 배열의 크기를 계산해보도록 하겠다.

텍스트이(가) 표시된 사진

자동 생성된 설명 

3x3x4=48이고, 7x9x4는 252이다. 따라서 sizeof 연산자가 2차원 배열의 크기를 정확히 계산해서 반환하고 있음을 실행결과를 통해서 확인할 수 있다.

- 2차원 배열요소의 접근

int arr[3][3]; 배열이 선언되어 모든 배열의 요소가 0으로 초기화되었다고 가정해보자

그렇다면 다음의 형태로 메모리 공간에 배열이 존재하게 된다.

텍스트, 전자기기, 계산기, 키보드이(가) 표시된 사진

자동 생성된 설명

이 상황에서 arr[0][0]=1;이 실행되면, 인덱스 값을 기준으로 세로 0, 가로 0의 위치에 1이 삽입되어 다음의 형태가 된다.

텍스트, 전자기기, 계산기, 키보드이(가) 표시된 사진

자동 생성된 설명

이어서 arr[0][1]=2;가 실행되면, 인덱스 값을 기준으로 세로 0, 가로 1의 위치에 2가 삽입되어 다음의 형태가 된다.

텍스트, 전자기기, 계산기, 키보드이(가) 표시된 사진

자동 생성된 설명

이어서 arr[2][1]=5;가 실행되면, 인덱스 값을 기준으로 세로 2, 가로 1의 위치에 5가 삽입되어 다음의 형태가 된다.

텍스트, 전자기기, 계산기, 키보드이(가) 표시된 사진

자동 생성된 설명

이렇듯 이름이 arr인 int형 배열을 대상으로 세로 N번째 위치(1부터 시작해서 N번째), 그리고 가로 M번째 위치(1부터 시작해서 M번째)에 지정된 값을 변경 및 참조하는 방법을 일반화하면 다음과 같다.

arr[N-1][M-1]=20; // 세로 N, 가로 M의 위치에 정수 20을 저장

printf(“%d”, arr[N-1][M-1]); // 세로 N, 가로 M의 위치에 저장된 값 출력

프로그램 사용자로부터 층별로 두 가구가 사는 4층짜리 빌라의 가구별 거주인원 수를 입력 받는 예제를 작성해보겠다. 층별로 두 가구가 사는 4층짜리 빌라이니, 가로가 2이고 세로가 4인 2차원 배열로 표현하기에 좋은 상황이다.

텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트, 전자기기, 키보드, 파일이(가) 표시된 사진

자동 생성된 설명

- 2차원 배열의 메모리상 할당의 형태

2차원 배열은 물리적으로도 2차원의 형태로 존재할까? 컴퓨터의 메모리는 2차원적 구조가 아니다. 이는 메모리의 주소 값을 통해서 알 수 있다. 우리가 사용하는 메모리의 주소 값은 다음과 같이 1차원적 구조이다.

0x1001번지, 0x1002번지, 0x1003번지, 0x1004번지, 0x1005번지 . . . .

만약에 2차원적 구조였다면 메모리의 주소 값은 다음의 형태를 띠었을 것이다.

0x12-0x24번지, 0x12-0x25번지, 0x12-0x26번지, 0x12-0x27번지 . . . .

0x13-0x24번지, 0x13-0x25번지, 0x13-0x26번지, 0x13-0x27번지 . . . .

0x14-0x24번지, 0x14-0x25번지, 0x14-0x26번지, 0x14-0x27번지 . . . .

. . . .

따라서 2차원 배열도 메모리상에는 1차원의 형태로 존재한다. 예를 들어서 0~5로 초기화된, 세로길이가 3이고 가로길이가 2인 int형 배열을 선언하면, 그 배열은 다음의 구조로 메모리상에 존재하게 된다.

텍스트, 계산기, 전자기기이(가) 표시된 사진

자동 생성된 설명

그럼 실제로 위 그림의 형태로 배열이 할당되는지 확인하기 위해서 2차원 배열요소의 주소 값을 출력해보겠다.

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

위의 실행결과를 통해서, 배열요소 별 주소 값은 int형 변수의 크기인 4바이크만큼만 차이가 남을 알 수 있다.

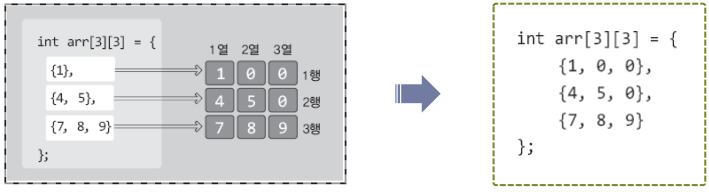
- 2차원 배열 선언과 동시에 초기화하기

1차월 배열과 마찬가지로 2차원 배열도 선언과 동시에 초기화가 가능하다. 2차원 배열의 가장 일반적인 초기화 방법은 다음 그림에서 보이는 바와 같다.

텍스트, 전자기기, 계산기이(가) 표시된 사진

자동 생성된 설명

위 그림에서 보이듯이 초기화 리스트 안에는 행 단위로 초기화할 값들을 별도의 중괄호로 명시해야 한다. 물론 모든 배열요소를 초기화해야 하는 것은 아니다. 다음과 같이 일부 요소에 대해서는 초기화를 생략할 수 있다. 그리고 이렇게 해서 비게 되는 공간은 1차원 배열의 경우와 마찬가지로 0으로 초기화된다.



그리고 마지막으로, 다음과 같이 하나의 중괄호 안에 초기화할 값을 순서대로 나열할 수도 있다.

텍스트, 전자기기, 스크린샷이(가) 표시된 사진

자동 생성된 설명

위 그림과 같이 행 단위 중괄호 없이 초기화 리스트를 나열하면 1행 1열부터 시작해서 배열의 마지막 요소까지 순서대로 값이 초기화되는데, 부족한 영역은 0으로 초기화된다. 그리고 위 그림의 배열선언은 다음과 같이 한 줄에 선언하는 것도 가능하다.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트, 검은색, 화면, 닫기이(가) 표시된 사진

자동 생성된 설명

- 배열의 크기를 알려주지 않고 초기화하기

1차원 배열선언 시, 초기화 리스트가 존재한다면, 배열의 길이를 명시하지 않아도 된다. 이는 2차원 배열에도 마찬가지이다. 2차원 배열도 초기화 리스트가 존재한다면, 그 길이를 명시하지 않아도 된다. 단, 여기에는 제약사항이 있다. 이와 관련해서 다음 문장을 보자.

int arr[][]={1, 2, 3, 4, 5, 6, 7, 8};

배열의 가로와 세로의 길이에 여러 가지 경우의 수가 존재하므로, 컴파일러가 판단할 수 없다.

하지만 하나만 가르쳐준다면,

int arr1[][4]={1, 2, 3, 4, 5, 6, 7, 8};

int arr2[][2]={1, 2, 3, 4, 5, 6, 7, 8};

이제 예측이 가능해졌다. arr1의 경우 가로의 길이가 4이니, 세로의 길이는 2가 되어야 한다. 그리고 arr2의 경우 가로의 길이가 2이니, 세로의 길이는 4가 되어야 한다. 이렇듯 2차원 배열을 선언과 동시에 초기화하는 겨우에는 배열의 세로길이만 생략이 가능하다.

**1-2. 3차원 배열**

- 3차원 배열의 논리적 구조

컨테이너, 건축자재, 상자, 광장이(가) 표시된 사진

자동 생성된 설명

위 그림의 큐빅퍼즐은 3차원의 구조를 띠고 있다. 가로의 길이와 세로의 길이, 그리고 높이가 3인 정육면체 구조이다. 3차원 배열은 중심부의 블록이 채워진 형태의 큐빅퍼즐에 비유할 수 있다. 앞서 우리는 가로와 세로로 이뤄진 2차원 배열을 공부하였는데, 3차원 배열은 여기에 높이의 개념이 추가되어 있다. 따라서 다음의 형태로 배열을 선언하게 된다.

int arr1[2][3][4]; // 높이 2, 세로 3, 가로 4인 int형 3차원 배열

double arr2[5][5][5]; // 높이, 세로, 가로가 모두 5인 double형 3차원 배열

그럼 sizeof 연산의 결과를 통해서 3차원 배열의 크기를 확인해보자.

텍스트이(가) 표시된 사진

자동 생성된 설명



2x3x4xsizeof(int)는 96이고, 5x5x5xsizeof(double)은 1000이다. 따라서 위의 출력결과를 통해서 3차원 배열이 논리적으로 직육면체 형태의 배열임을 이해할 수 있다.

- 3차원 배열의 선언과 접근

높이 정보의 반영을 제외하면 3차원 배열의 선언 및 접근방법은 2차원 배열의 선언 및 접근방법과 차이를 보이지 않는다. 다음 예제에서는 어느 고등학교 3학년 전체 학생의 국어와 수학점수를 반별로 구분하여 저장한 예를 보여준다. 각 반별 학생수는 3명이고 학급의 수도 3개라고 가정하였다.

텍스트, 화면, 손, 스크린샷이(가) 표시된 사진

자동 생성된 설명텍스트, 전자기기, 화면이(가) 표시된 사진

자동 생성된 설명

2. 시간 함수

텍스트, 실내, 텔레비전, 모니터이(가) 표시된 사진

자동 생성된 설명

#include <unistd.h> 헤더 파일은 sleep( ) 함수를 사용하기 위해, <time.h>는 time( ) 함수를 사용하기 위해 선언해야 한다.

endTime에 (unsigned)time(NULL) 함수를 넣는다. (unsigned)부호 비트가 0으로 시간 값을 음수가 아닌 양수로만 나타내기 위해서이다. 그럼 1970년 1월 1일부터 지금까지에 시간을 초로 반환해준다. 이 값에 6을 더해준다. 이거로 원하는 시간(6초)까지 초를 제한할 것이다.

그리고 while문을 통해 startTime에 현재 시간을 넣어주면 반복을 통해 숫자가 초마다 늘어난다. 그리고 조건문을 통해 0초가 되면 종료되게 코드를 완성했다. 확인을 해보면,



이렇게 1초마다 줄어들면서 0초까지 반복되고 종료되는 것을 확인할 수 있다. 그런데 초가 계속 나와 불편하다.

텍스트, 실내, 화면, 스크린샷이(가) 표시된 사진

자동 생성된 설명

 sleep(0.01)로 잠시 지연을 시키고 system(“clear”)로 화면을 지우면 된다.

3. 헤더파일

**#include <stdlib.h>**

|  |  |
| --- | --- |
| int **rand** ( void ); | 0부터 RAND\_MAX 사이의 의사 난수를 반환한다. |
| void **srand** ( unsigned int seed ); | 의사 난수 발생기를 seed로 초기화한다. 보통 seed는 time(NULL)로 설정된다. |

**#include <time.h>**

C언어의 표준 함수를 사용하여 컴퓨터의 시간을 가져온다. 시간은 단순한 수치로 처리되어 있기 때문에, 날짜로 취급하려면 시스템의 로컬 시간으로 변환해야 한다.

- 시간을 처리하기

데이터 관리를 할 때, 시간은 중요한 존재이다. 예를 들어 정보가 업데이트 되었을 때는 프로그램 업데이트 로그를 남기는 등의 처리가 요구된다. 프로그램은 지금 날짜를 알고 싶다면, 시스템 시계에서 시간을 산출한다.

현재 시간을 얻거나, 새로 설정하는 프로그램을 만드는 경우 기본적으로 시스템에 문의해야 한다. 시스템이 지원하고 있는 시간 관련 기능을 최대한으로 발휘시키는 경우는 C언어의 표준 함수에서만 지원할 수 있는 것이 아니기 때문이다. 그러나, 복잡한 API를 사용하지 않아도 시간의 취득이나 변환 등의 기본적인 처리는 표준 함수에서 지원되고 있다.

시간 관련 함수는 time.h 헤더 파일에 선언되어 있다. 시스템 시계에서 시간을 얻으려면 time() 함수를 사용한다.

- time() 함수

**time\_t** time( **time\_t** \*timer );

timer에는 시스템 시간을 얻는 time\_t형 변수의 포인터를 지정한다. time\_t형은 time.h 헤더 파일에 정의되어 있는 시간을 나타내는 산술형의 typedef 명이다. time\_t형의 실제 형식은 구현에 따라 달라진다. time() 함수는 인수에서 얻은 값과 같은 값을 반환한다. 즉, time() 함수는 시간을 얻는 수단으로 포인터에 간접 참조에서 값을 저장하는 방법 및 반환 값에서 얻는 방법을 선택할 수 있다. 반환 값을 얻는 경우는 인수를 NULL로 해도 상관없다.

이 함수에서 얻을 수 있는 값은 시스템의 부호화 된 time\_t형 달력 시간이다. 시간을 얻을 수 없는 경우는 -1을 돌려준다. time() 함수가 반환하는 값은 구현에 의존하기 때문에, 사용하는 시스템의 시간 방식을 확인한다. Microsoft Windows에는 시스템 클럭에 따라 만국 표준시(UCT)의 1970년 1월 1일 0시 0분 0초부터 경과된 시간을 초단위로 나타내는 숫자를 반환한다.

이대로는 매우 시간이라고는 생각되지 않는듯한 수치가 나타날 뿐이기 때문에, 이를 현재 시간으로 변환해야 한다. 연산에 의해 경과 시간에서 현재 시간을 얻을 수 있지만, localtime() 함수가 이 처리를 수행해 준다.

- localtime() 함수

struct tm \***localtime**( **const** **time\_t** \*timer );

timer에는 변환하는 시간 값에 대한 포인터를 지정한다. 시간은 년/월/일 등의 값을 나타내는 멤버가 있는 tm 구조체에 대한 포인터를 반환한다. tm 구조체는 다음과 같이 선언되어 있다.

- tm 구조체

**struct** **tm** {

**int** tm\_sec; /\* 초 - [0～61] （閏秒を考慮） \*/

**int** tm\_min; /\* 분 - [0～59] \*/

**int** tm\_hour; /\* 시 - [0～23] \*/

**int** tm\_mday; /\* 일 - [1～31] \*/

**int** tm\_mon; /\* 월 - [0～11] \*/

**int** tm\_year; /\* 1900부터의 년 \*/

**int** tm\_wday; /\* 일요일부터의 요일 - [0～6] \*/

**int** tm\_yday; /\* 년초부터의 통산 일수 - [0～365] \*/

**int** tm\_isdst; /\* 서머 타임이 유효하면 양수, 유효하지 않으면 0, 불명이면 음수\*/

};

localtime() 함수는 인수 timer에서 지정된 시간 값을 바탕으로 각 멤버를 적절한 값으로 초기화되어 있는 tm 구조체에 대한 포인터를 반환한다. 이 함수가 반환된 포인터를 이용하면 시간 값에서 사람이 이해할 수 있는 시간을 표시하는 것이 가능하다.

tm 구조체의 월은 0부터 시작되므로, 1월은 0임을 주의한다. 또한 년은 1900년부터 계산된 값이다. 실제 서기로 변환하려면 1900을 추가해야 한다.

#**include** <stdio.h>

#**include** <time.h>

**int** **main**() {

**struct** **tm** \***date**;

**const** **time\_t** t = time(NULL);

date = localtime(&t);

printf(

"%d/%d/%d %d:%d:%d\n" , date->tm\_year + 1900 , date->tm\_mon + 1 ,

date->tm\_mday , date->tm\_hour , date->tm\_min , date->tm\_sec

);

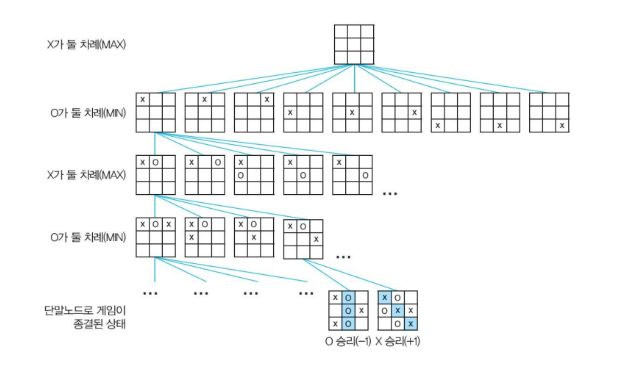
**return** 0;

}

위 코드는 현재 시간을 표시하는 프로그램이다. 시간은 년/월/일 시:분:초 형태로 표준 출력에 표시된다. 프로그램은 먼저 time() 함수를 사용하여 시스템 시계에서 시간 값을 가져온다. 다음에 localtime() 함수를 사용하여 시간 값을 tm 구조체로 변환한다. 다음은 tm 구조체의 시간을 나타내는 각 멤버에 엑세스하여 시간을 표시한다.

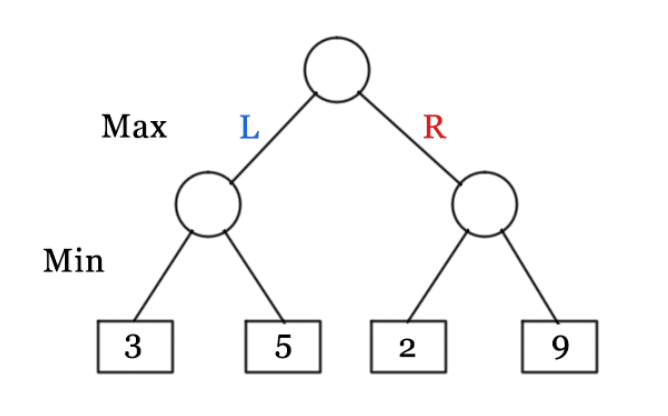
4. 미니맥스 알고리즘

- 틱택토 게임트리



틱택토 게임 트리의 일부이다. 틱택토 게임 트리의 크기는 게임보드가 3x3 크기를 가지고 있고, 한 곳에 수를 놓으면 다른 사람이 놓을 수 있는 곳이 하나 줄어들게 된다. 따라서 9! = 362,880이 된다. 그렇다면 362,880의 크기를 전부 계산할까? 결론부터 이야기하자면, 그런 경우도 존재할 수 있지만 그러기 힘들다.

- 미니맥스(minimax) 알고리즘



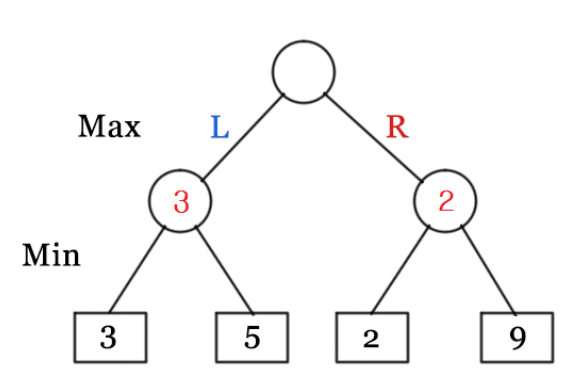
틱택토를 해결하기 위한 미니맥스 알고리즘을 실행할 때 보드의 모든 미래 가능한 상태를 시각화하여 작동하고 이를 트리 형태로 구성한다. 현재 보드 상태가 알고리즘(트리의 루트)에 주어질 때, n가지 (여기서 n은 AI에 의해 선택될 수 있는 이동의수 / AI가 될 수 있는 빈 셀의 수를 나타냄)로 분할된다. 이러한 세 상태 중 하나가 최종 상태인 경우 이 상태에 대해 더 이상 분할이 수행되지 않고 다음과 같은 방식으로 점수가 할당된다.

점수 = +1 (AI가 이기는 경우)

점수 = -1 (AI가 지는 경우)

점수 = 0 (무승부가 발생하는 경우)

쉽게 말해 상대방이 항상 최선의 수를 둔다는 가정하에, 보드의 다음 가능한 형태를 트리형태로 만들고 선택한다.



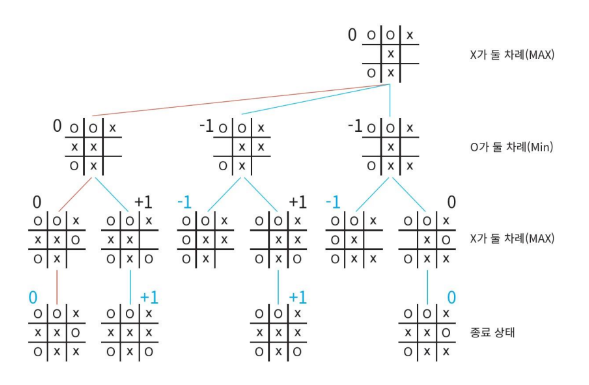
MIN은 숫자가 작을수록 이기고 MAX는 숫자가 클수록 이길 확률이 높다. 따라서 아래에서부터 보면 MIN은 작은 숫자를 고르기 때문에 3과 2를 선택한다.

텍스트, 손목시계이(가) 표시된 사진

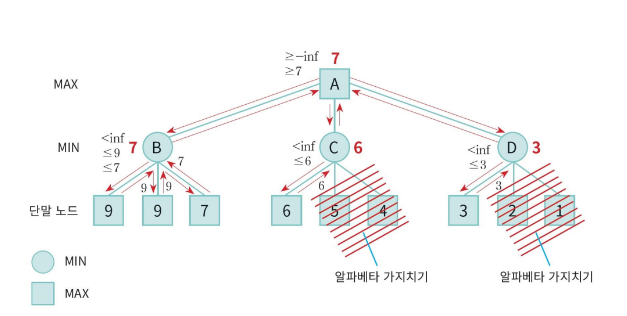
자동 생성된 설명

MAX는 큰 수를 고르기 때문에 2와 3중 3을 선택한다.

- 틱택토 게임에서의 미니맥스 예시



- 알바베타 가지치기



미니맥스 알고리즘에서 형성되는 탐색 트리 중에서 상당 부분은 결과에 영향을 주지 않으면서 가지들을 쳐낼 수 있다. 이것을 알파베타 가지치기라고 한다. 위 예시를 들어 설명하면 MAX는 -∞를 초기값으로, MIN은 +∞를 초기값으로 설정하고 노드를 탐색한다. A -> B로 갈 때 B는 초기값 +∞와 9중에서 MIN(+∞,9)를 하여 최소값 9를 찾는다. 이후 MIN(9,9) = 9 -> MIN(9,7) = 7을 계산하여 최소값 7을 찾는다. 그리고 B의 단말 노드의 계산을 마친 후 A는 초기값 -∞와 B에서 찾은 최소값 7과 MAX(-∞,7)을 계산하여 최대값 7을 찾는다.

여기서부터가 중요하다. 이제 A->C로 경로를 탐색하는데 이전 연산과 마찬가지로 MIN(+∞,6) = 6을 계산한다. 이후 C의 단말노드 5와 4는 계산해 볼 필요가 없다. 왜냐하면 A는 현재 가지고 있는 숫자 7보다 더 큰 숫자를 찾고 있는데 C는 6보다 작은 수를 찾아 보낼 것이기 때문이다. A->D도 위와 같다. 따라서 부모의 현재 상태에 따라서 트리를 전부 탐색할 필요는 없다. 미니맥스 알고리즘에서 형성되는 탐색 트리 중에서 상당부분은 결과에 영향을 주지 않으면서 가지들을 쳐낼 수 있다. 이것을 알파베타 가지치기라고 한다. 탐색을 할 때 알파 값과 베타 값을 비교하여 불필요한 탐색을 중지할 수 있다. MAX는 알파 값만을 업데이트하고 MIN은 베타 값만 업데이트 한다.

5. 난수

C언어에서 난수를 만들기 위해서는 rand()라는 함수를 사용하면 된다. rand() 함수는 stdlib.h 헤더파일에 포함되어 있다.

텍스트, 실내, 군중이(가) 표시된 사진

자동 생성된 설명

8번째 줄에 random = rand()%9; 라는 코드가 있다. 해석하면 rand() 함수에 의해 난수를 생성하고 그 숫자를 9로 나눈 나머지를 random 변수에 대입하는 코드이다.

rand() 함수에 의해 생성되는 난수: 0 ~ 32767

rand() 함수에 의해 생성된 난수를 9로 나눈 나머지(%)의 값은 0부터 8까지이다.

이를 1부터 9까지 난수를 생성하고 싶을 때는 random = rand()%9 + 1; 로 변경하면 된다. 이는 0부터 8까지 반환되는 난수에 1을 더해줌으로써 1부터 9까지 반환할 수 있도록 하는 코드이다.

하지만 여기서 문제점이 발생한다. 분명 개발자가 원하는 난수를 생성하고 출력했지만 그 패턴이 매번 일정하다. 다시말해, 프로그램 실행 시 항상 같은 난수가 생성된다. 이는 진정한 난수라고 말할 수 없다. 그래서 srand()라는 함수를 사용해야한다. rand() 함수를 사용하기 전에 srand(time(NULL)); 라는 코드를 추가해야한다. 무슨뜻이냐면 srand() 함수는 rand()라는 함수에 무작위의 시드 값을 주기 위한 함수이며 그 파라미터로 time(NULL)이라는 매개변수를 전달한다. time(NULL)은 1970년 1월 1일 이후 경과된 시간을 초 단위로 반환하는 함수이다. 이로써 1초 단위로 매번 다른 시드값을 생성해 rand() 함수를 호출하는 것이다.

텍스트, 실내, 화면, 군중이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명