

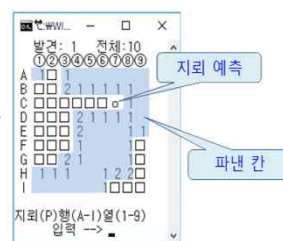
게임으로 배우는 C++

05 CHAPTER

함수의 진화



지뢰제거 게임(Mine Sweeper)



• Mine Sweeper

- 매설된 지뢰를 터트리지 않고 모두 안전하게 찾아내는 게임
- 지뢰가 없는 칸에는 인접한 8칸의 지뢰 수가 저장
- 지뢰 맵: 이차원 배열
- 지뢰의 매설: 난수
- 인접한 8칸에 지뢰가 하나도 없으면→연쇄적 파내기→재귀 호출
- 전체적으로 꽤나 복잡한 게임

5장 학습 목표



- 포인터의 개념과 활용 능력을 기른다.
- 배열과 포인터의 관계와 포인터 연산의 개념을 이해한다.
- 주소(값)에 의한 호출의 개념을 이해하고 활용할 수 있는 능력을 기른다.
- 참조자의 개념을 이해한다.
- 참조에 의한 호출이 값에 의한 호출과 어떻게 다른지를 이해하고 활용할 수 있는 능력을 기른다.
- 순환호출의 개념을 이해하고 게임 구현에 활용해 본다.

3

5.1 포인터



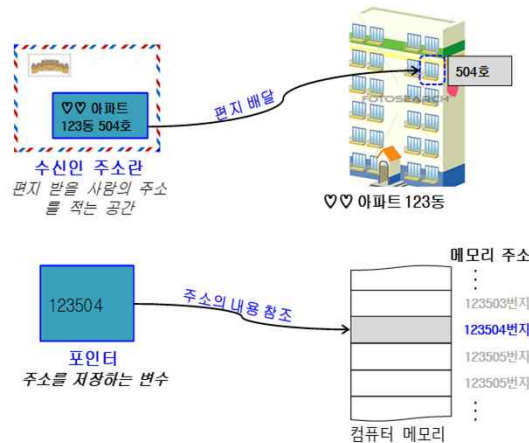
- 포인터(Pointer)란?
- 포인터의 선언과 활용
- 다중 포인터
- 포인터 연산
- 포인터와 배열, 포인터와 구조체
- 포인터 초기화

4

포인터(Pointer)란?



- 포인터 변수 또는 포인터
 - 메모리 주소를 저장하기 위해 사용되는 변수

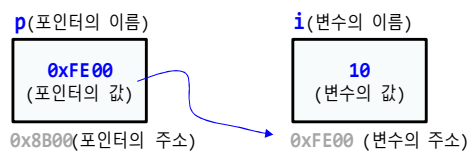


5

포인터 선언



```
int i = 10;
int* p;
p = &i;
```



- 포인터 변수 선언

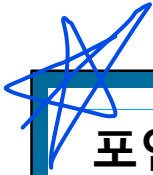
```
int* pi;    // int 변수를 가리킬 목적의 포인터 pi를 선언
float* pf;  // float 변수를 가리킬 목적의 포인터 pf를 선언
```

- 여러 개의 변수 선언

```
char* p, q, r;    // p는 char* 변수, q와 r은 char변수
char *p, *q, *r;  // p, q, r 모두 char*형 변수
```

- 사용하기 전에 반드시 초기화되어야 함

6



포인터 활용



```
int i = 10;  
int* p;  
p = &i;
```

p(포인터의 이름)
0xFE00
(포인터의 값)
0x8B00(포인터의 주소)

i(변수의 이름)
10
(변수의 값)
0xFE00 (변수의 주소)

```
*pi = 20; // i=20과 동일
```

• * 연산자 사용의 여러가지 예

```
사용예#1: c = a * b;  
사용예#2: int * p;  
사용예#3: *p = 20;
```

& 연산자

변수의 주소를 추출

* 연산자

포인터가 가리키는 곳의 내용을 추출

• 비트단위 AND 연산자 &

```
사용예#1: c = a & b;  
사용예#2: p = &i;  
사용예#3: int & p; // 참조자 (C++에서 추가됨)
```

↳ Err

7

int & p = 20; LOJ

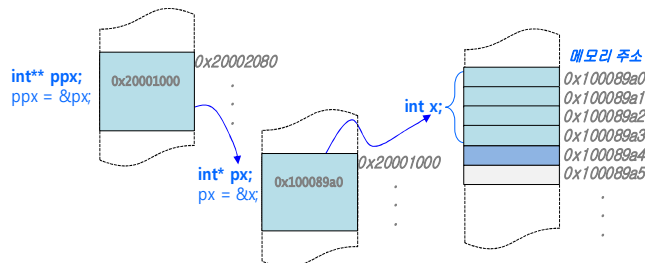
다중 포인터




• 포인터 변수의 값이 어떤 변수의 주소가 아니라 다른 포인터 변수의 주소인 경우

```
int x;  
int* px = &x;  
int** ppx = &px;
```

표현	자료형	동일한 표현
x	int	*px, **ppx
px	int*	*ppx, &x
ppx	int**	&px



8



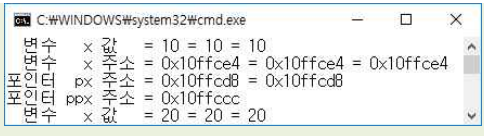
```

void main()
{
    int x = 10;
    int* px = &x;
    int** ppx = &px;


    printf(" 변수 x 값 = %d = %d = %d\n", x, *px, **ppx);
    printf(" 변수 x 주소 = 0x%x = 0x%x = 0x%x\n", &x, px, *ppx);
    printf("포인터 px 주소 = 0x%x = 0x%x\n", &px, ppx);
    printf("포인터 ppx 주소 = 0x%x\n", &ppx);

    *px = 20;
    printf(" 변수 x 값 = %d = %d = %d\n", x, *px, **ppx);
}

```



9



포인터 연산

- 포인터에 대한 사칙연산
 - 포인터가 가리키는 객체단위로 계산된다.

```

int A[5], int *p;
p = &A[2];
p-1;
p+1;

```

포인터 변수 p

0x10001008

int A[5];

A[0]	0x10001000
A[1]	0x10001004
A[2]	0x10001008
A[3]	0x1000100c
A[4]	0x10001010
...	

p-1

p+2

10

포인터와 배열



- 배열의 이름: 사실상의 포인터와 같은 역할
 - 컴파일러가 배열의 이름을 배열의 첫 번째 주소로 대치

```
void main()
{
    int list[5] = {1,2,3,4,5 };
    int* p = list;

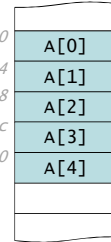
    printf("list= ");
    for (int i = 0; i < 5; i++)
        printf("%d ", *p++);

    p = list;
    for (int i = 0; i < 5; i++)
        sum += p[i];
    printf("\nsum = %d\n", sum);
}
```

배열과 포인터
`int A[5];`

A

0x1000
0x1004
0x1008
0x100c
0x1010



```
list= 1 2 3 4 5
sum = 15
```

11

포인터와 구조체



```
typedef struct {
    int degree;
    float coef[MAX_DEGREE];
} Polynomial ;
```

```
Polynomial a;
Polynomial* p;
p = &a;
a.degree = 5;
p->coef[0] = 1;
```

- 구조체 멤버의 접근
 - 구조체에서는 "." 연산자
 - 포인터에서는 "->" 연산자

구조체를 이용한 표현		포인터를 이용한 표현	
a.degree	(&a)->degree	p->degree	(*p).degree
a.coef[0]	(&a)->coef[0]	p->coef[0]	(*p).coef[0]

12

포인터 초기화



• 포인터 사용시 주의점

- ① 포인터가 값을 가리키고 있지 않을 때는 NULL로 설정

```
int *p = NULL;
```

- ② 초기화가 안 된 포인터가 가리키는 곳 사용 절대 금지

```
int *px;      // 포인터 pc는 초기화가 안 되어 있음
*px = 10;     // 매우 위험한 코드
```

- ③ 포인터 타입 간에 반드시 명시적인 형 변환 사용

13

5.2 주소에 의한 호출



- 함수 호출과 포인터
- "주소" 값에 의한 호출
- 주소를 반환하는 함수?

14

함수 호출과 포인터



- 배열에서 최댓값과 최솟값을 찾는 함수

```
void findMinMax(int* a, int len, int* pmin, int* pmax)
{
    if (pmin != NULL) {
        *pmin = a[0];
        for (int i = 1; i < len; i++)
            if (*pmin > a[i]) *pmin = a[i];
    }
    if (pmax != NULL) {
        *pmax = a[0];
        for (int i = 1; i < len; i++)
            if (*pmax < a[i]) *pmax = a[i];
    }
}
...
findMinMax(arr, 10, &min, &max);
findMinMax(arr, 10, &min, NULL);
findMinMax(arr, 10, NULL, &max);
...
```

두개 반환

15

“주소” 값에 의한 호출



- 복소수의 초기화 함수 문제
 - 값에 의한 호출: 인수 값이 매개변수로 복사됨

```
void resetComplex(Complex a) { a.real = a.imag = 0.0; }
```

- 해결방법 → 주소 값에 의한 호출
 - 매개 변수의 자료형을 포인터형으로 변환

```
void resetComplex(Complex* a) { a->real = a->imag = 0.0; }
void main()
{
    Complex c;
    resetComplex(&c);
    printComplex(c, "reset(c)=");
}
```

16

“주소” 값에 의한 호출



• 복사되는 내용: 인수 → 매개변수

- 값에 의한 호출: “복소수 객체” 값
- 주소 값에 의한 호출: “복소수 객체의 주소” 값

```
9. void resetComplex(Complex* p) { p->real = p->imag = 0.0; }
```

```
...
```

<“주소”값에 의한 호출>
인수 c의 주소가 p에 복사됨.
포인터 p는 c를 가리킴.

포인터 p c의 주소

p c의 주소

p가 가리키는 복소수(c)의 real과
imag 멤버를 0으로 변경함.
실제로는 c의 멤버가 변경됨.

```
14. resetComplex(&c);
```

```
15. printComplex(c, "reset(c)=");
```

함수 종료시 포인터 p는 소멸됨.
반환 후 c가 변경되어 있음.

17

주소를 반환하는 함수?



• 배열을 역순으로 복사하는 함수 구현

```
void reverseArray(const int a[], int b[], int len) {...} // OK
```

```
int* reverseArray1(const int a[], int len)
```

```
int b[100];
```

```
for (int i = 0; i < len; i++)
```

```
b[len-i-1] = a[i];
```

```
return b;
```

// 잘못된 구현 방법

```
}
```

```
int* reverseArray2(const int a[], int len) {
```

```
static int b[100];
```

```
for (int i = 0; i < len; i++)
```

```
b[len-i-1] = a[i];
```

```
return b;
```

// 여전히 문제가 있는 방법

```
}
```

```
C:\WINDOWS\system32\cmd.exe
배열 a: 3 24 82 12 34 7 53 17 26 51
배열 b: 51 26 17 53 7 34 12 82 24 3
배열 c: 943481294345967647232 1 1 1 3262473320163256889434564
배열 d: 51 26 17 53 7 34 12 82 24 3
```

18

5.3 참조형



- 참조형(Reference type)이란?
- 왜 사용할까?

19

참조형(Reference type)



- 기존의 메모리 공간에 별명을 붙이는 방법

```
int var = 10;    // int형 변수 var을 선언하고 10으로 초기화하는 문장
int& ref = var;  // 변수 var의 별명인 참조 변수 ref를 선언
ref = 20;        // var의 별명 ref에 20을 복사했으므로 val이 20이 됨
```

- 선언과 동시에 반드시 초기화해야 함

```
int& ref;        // 컴파일 오류: 별명은 선언과 동시에 초기화되어야 함
ref = var;       // 컴파일 오류: 선언 이후에 변경할 수 없음
```

- 사용 분야

- 기존 변수들을 보다 간편하게 사용하기 위해
- 함수의 매개 변수로 사용하기 위해
- 함수의 반환형으로 사용하기 위해

20

왜 사용할까?



- 기존 변수들을 보다 간편하게 사용하기 위해

```
// 2차원 배열의 항목에 대한 참조
int map[3][3] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
int sum = 0;
for (int i = 0; i < 3; i++)
for (int j = 0; j < 3; j++) {
    int& elem = map[i][j];           // map[i][j]에 대한 별명 elem
    elem = i * 3 + j + 1;           // map[i][j] = ...;
    sum += elem;                     // elem += map[i][j];
    printf(" %d", elem);            // map[i][j]를 출력
}

PlayInfo teamKorea[4] = { { "김현수", 20, 20.0 },
                          { "오승환", 30, 30.0 }, { "황재균", 40, 40.0 } };
PlayInfo& play = teamKorea[2];
printf("%s %d %4.1f\n", play.name, play.nMove, play.tElapsed);
```

21

5.4 참조에 의한 호출



- 참조에 의한 호출
- 참조형과 함수의 반환

22

참조에 의한 호출



- 값에 의한 전달과 완전히 다름
 - (인수)값을 (매개변수에) 복사하지 않고 별명을 전달함
 - 매개변수는 인수의 별명이 됨. (인수는 반드시 변수)

```
void findMinMax(int* a, int len, int& min, int& max) {
    min = a[0];
    for (int i = 1; i < len; i++)
        if (min > a[i]) min = a[i];

    max = a[0];
    for (int i = 1; i < len; i++)
        if (max < a[i]) max = a[i];
}

void main() {
    int arr[10] = { 3, 24, 82, 12, 34, 7, 53, 17, 26, 51 };
    int x, y;
    findMinMax(arr, 10, x, y);
    printf("최소~최대: %2d~ %2d\n", x, y);
}
```

C. — □ ×
최소~최대: 3~82

23

참조에 의한 호출



```
9. void resetComplex(Complex& a) { a.real = a.imag = 0.0; }
```

...

참조자 a=c

a=c

<참조에 의한 호출>
매개변수 a는 복소수 c의 별명.
따라서 a는 c와 동일한 변수.

a(실제로는 c)의 real과 imag 멤버
를 0으로 변경함. c가 변경됨.

c의 주소

```
14. resetComplex(c);
15. printComplex(c, "reset(c)=");
```

함수 종료시 소멸될 것이 없음. a는
원래 새로운 변수가 아니었음.
반환 후 c가 변경되어 있음.

C:\WINDOWS\system32... — □ ×
reset(c)= 0.00 + 0.00i

24

참조형과 함수의 반환



- 함수 호출 횟수 반환 + **횟수 리셋** 기능

```
int getCount() {  
    static int count = 0;  
    count++;  
    ...  
    return count;  
}
```

```
int getCount(bool bReset) {  
    static int count = 0;  
    if(bReset) count=0;  
    count++;  
    ...  
    return count;  
}
```

```
static int count = 0;  
int getCount() {  
    count++;  
    ...  
    return count;  
}  
...  
count=0;
```

```
int& getCount() {  
    static int count = 0;  
    count++;  
    ...  
    return count;  
}  
...  
getCount() = 0;
```

25

5.5 심화학습: 재귀 함수



- 재귀 함수 (recursive function)란?
- 하노이탑 문제
- 영역 채색 문제

26

재귀 함수 (recursive function)



- 본질적으로 순환적인 문제에 적합
 - 순환

```
int factorial(int n) {
    if( n == 1 ) return 1;
    else return (n * factorial(n-1) );
}
```

- 반복으로 변경 가능

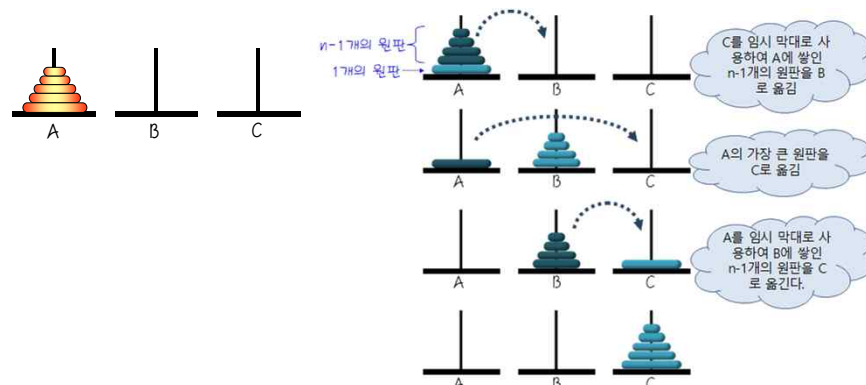
```
int factorialIter(int n) {
    int result=1;
    for( int k=n ; k>0 ; k-- )
        result = result * k;
    return result;
}
```

27

하노이탑 문제



- A에 쌓인 n개의 원판을 B를 이용해 C로 옮기는 문제

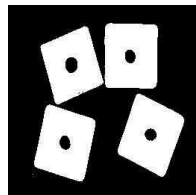


28

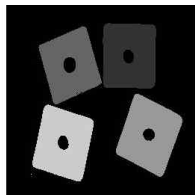
영역 채색 문제



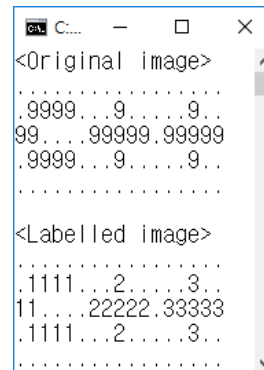
- 영역 채색(blob coloring), 또는 연결화소분석법 (connected component labelling)



(a) 이진 영상



(b) 영역 채색 결과: 연결된 객체가 같은 색으로 채색됨



29

5.6 심화응용: 지뢰 찾기 게임



- 분석 및 설계
- 구현
- 고찰

30

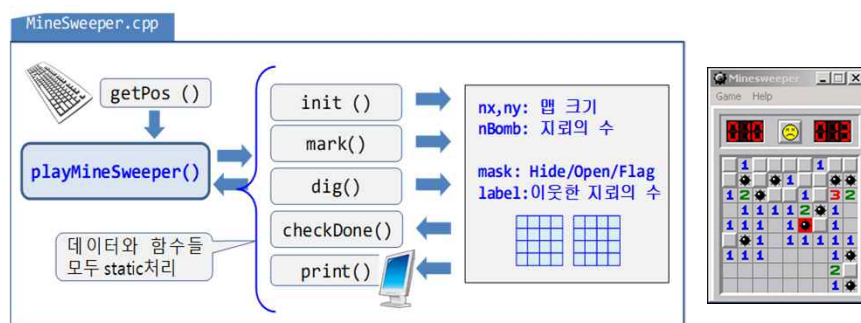
응용: 지뢰 찾기 게임



- 게임 조건
 - Width x Height 의 이차원 공간에 지뢰가 매설됨
 - 사용자가 위치 선택: 안전(dig), 지뢰(flag)
 - 안전(dig) 위치에 지뢰가 있으면 게임은 실패로 종료
 - 아니면 숫자가 나타남(인접한 8칸에 있는 지뢰의 수)
 - 숫자가 0이면 인접한 8칸을 모두 자동으로 팜
 - 숫자들을 근거로 숨어있는 모든 지뢰를 찾음
 - 열지 않은 칸+깃발의 합==지뢰 수: 성공 종료
 - 첫 번째 선택 → 운이 없으면 지뢰를 열 수 있음.
 - 지뢰의 매설은 난수를 발생하여 처리.

31

게임 처리 과정



```
#define DIM 9
enum LabelType { Empty = 0, Bomb = 9 };
enum MaskType { Hide = 0, Open, Flag };
static int MineMapMask[DIM][DIM]; // Hide, Open, Flag
static int MineMapLabel[DIM][DIM]; // 0~8, 9(Bomb)
static int nx=DIM, ny=DIM;
static int nBomb=DIM;
```

32

함수들



```

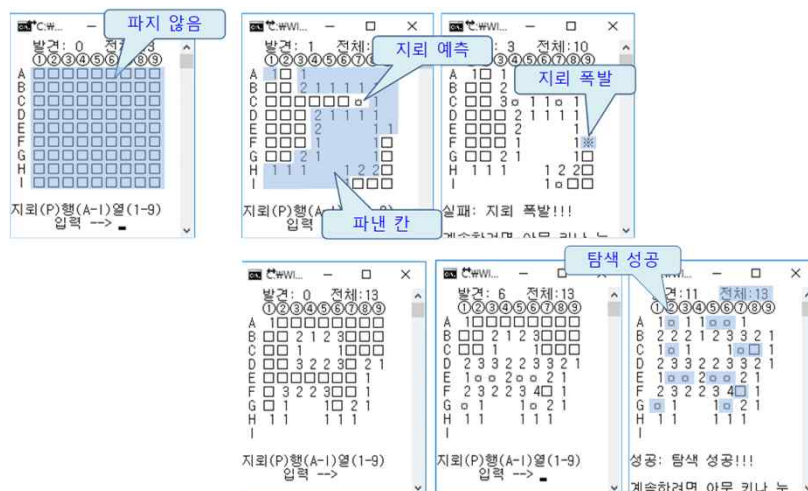
inline int& mask(int x, int y) { return MineMapMask[y][x]; }
inline int& label(int x, int y) { return MineMapLabel[y][x]; }
inline bool isValid(int x, int y){ return (x>=0&&x<nx &&y>=0&&y<ny); }
inline bool isBomb(int x, int y) { return isValid(x,y)&&label(x,y)==Bomb; }
inline bool isEmpty(int x, int y){ return isValid(x,y)&&label(x,y)==Empty; }

static void dig(int x, int y) {...} // (x,y)를 파는(여는) 함수
static void mark(int x, int y){...} // (x,y)에 깃발을 꽂는 함수
static int getBombCount(){...} // 깃발의 수를 계산하는 함수
static void print(){...} // 지뢰 맵의 화면 출력 함수
static int countNbrBombs(int x, int y){...} // 인접한 지뢰의 수 계산 함수
static void init( int total = 9 ) {...}
static bool getPos(int& x, int& y) {...} // 키보드 좌표 입력 함수
static int checkDone(){...} // 게임 종료 검사 함수

void playMineSweeper(int total){...} // 지뢰 찾기 주 함수
    
```

33

게임 결과 예



34

고찰



- 키보드로 좌표를 입력하는 불편 → 윈도우 프로그래밍 + 마우스 이벤트
- 맵의 크기를 변경 → 동적 할당
- 두 자릿수(10) 이상 좌표 입력 → 행과 열 번호 이용
- 사용된 주요 문법
 - 참조형 매개 변수와 참조자의 반환.
 - 인라인 함수와 디폴트 매개변수.
 - 재귀 호출 (어려운 개념임. 책의 다른 코드에는 사용되지 않음.)
 - 정적 함수와 정적 전역 변수
 - 나열형 enum과 bool 반환 함수 및 문자 처리 함수 toupper()

35

5장 요약문제, 연습문제, 실습문제



36



감사합니다!