



2.C 기반의 C++ part 2

❖ bool 자료형

❖ 레퍼런스 (reference)

❖ new & delete

Jong Hyuk Park

들어가기에 앞서서



❖ 문제 1 : `const` 키워드의 의미

- `const int n=10;`
- `const int* n;`
- `int* const n;`
- `const int* const n`

❖ 문제 2 : `stack`의 용도 및 특징

❖ 문제 3 : `heap`의 용도 및 특징

❖ 문제 4 : `malloc` & `free` 함수의 필요성

들어가기에 앞서서



❖ 문제 1 : const 키워드의 의미

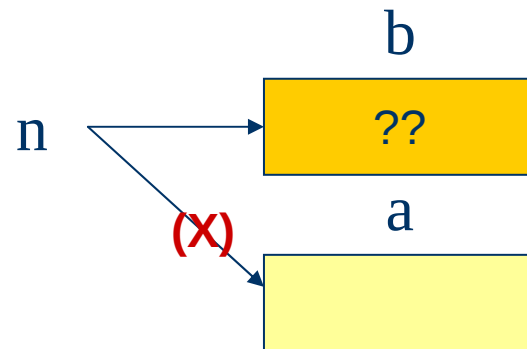
- 변수를 상수화하기 위해 사용

▪ `const int n=10;` → `n=12` (X)

▪ `const int* n;` // 데이터 상수화

ex) `n` → 12 `*n = 20` (X)

▪ `int* const n;` // 포인터 상수화



`n=&a` (X)

▪ `const int* const n` // 데이터 상수화 포인터 상수화

들어가기에 앞서서



- ❖ 문제 2 : stack의 용도 및 특징
- ❖ 문제 3 : heap의 용도 및 특징
- ❖ 문제 4 : malloc & free 함수의 필요성

들어가기에 앞서서



```
void function (int);
```

```
int main(void)
```

```
{
```

```
    int size;
```

```
    cin>>size;
```

```
    function(size);
```

```
    return 0;
```

```
}
```

```
void function(int i)
```

```
{
```

```
    int array[i];
```

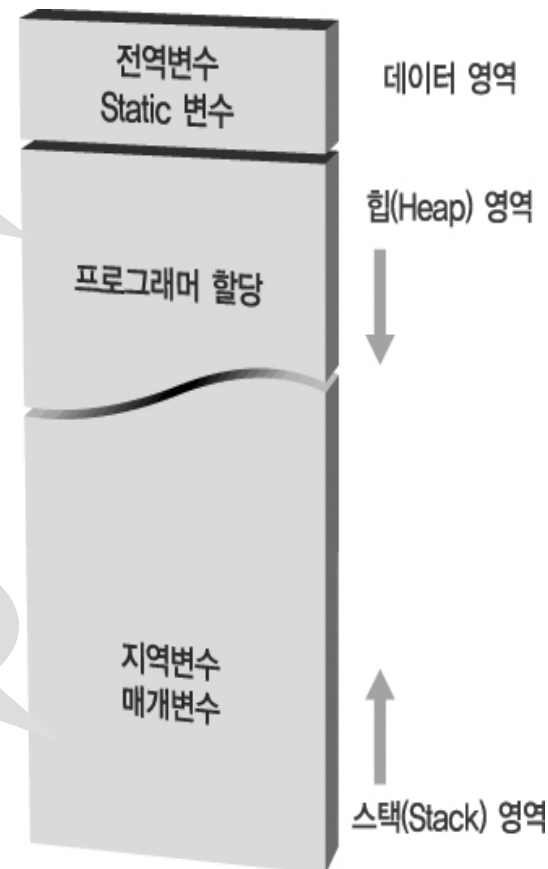
```
}
```

문제점?

런 타임
크기 결정

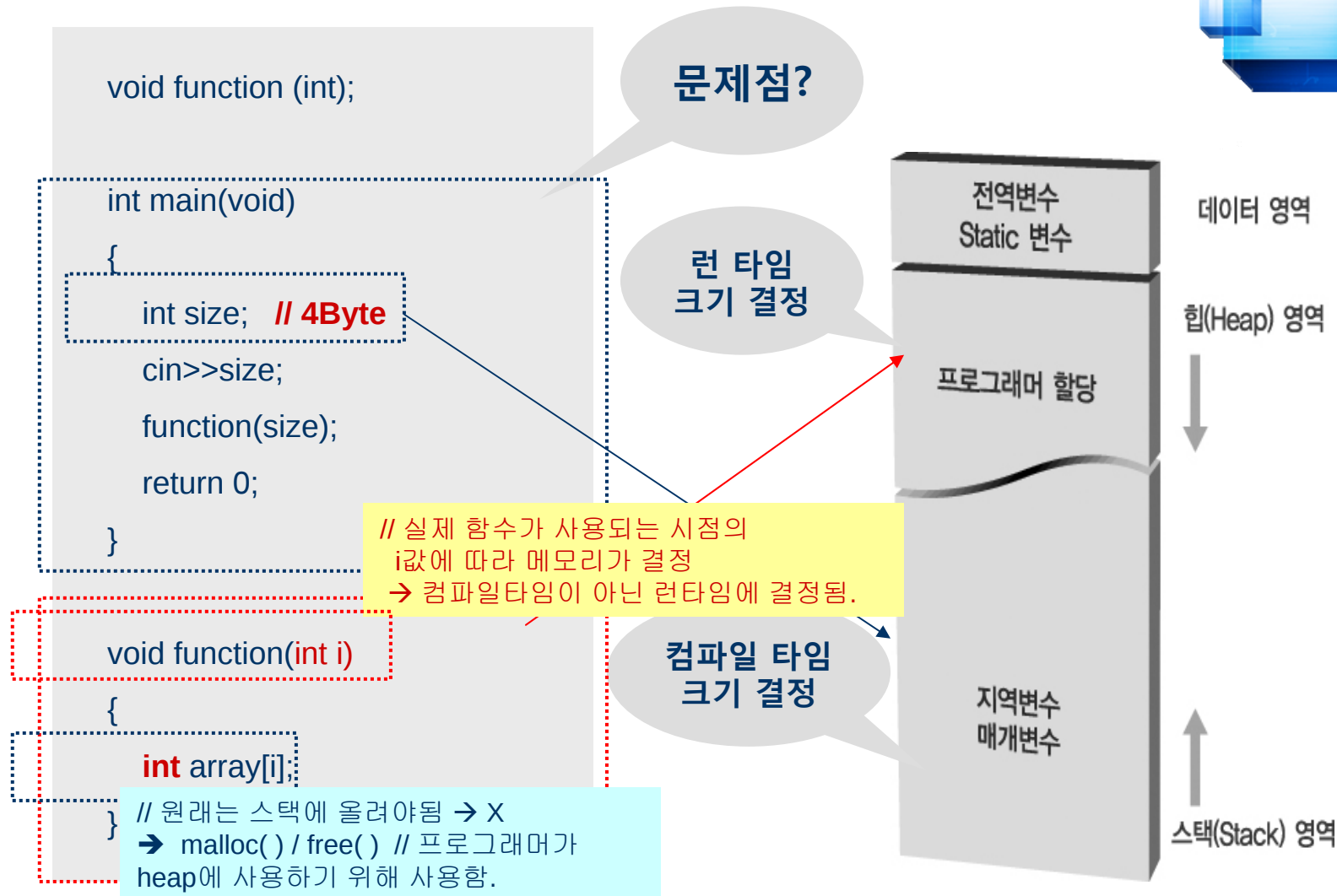
컴파일 타임
크기 결정

- 전역변수는 프로그램 시작부터 종료까지 메모리공간을 차지함.
- Static 변수도 동일하나, 초기화되는 시점이 사용된 함수가 Call되는 시점임.



C++ Programming

들어가기에 앞서서



C++ Programming



bool 자료형

Jong Hyuk Park

Bool 자료형



❖ 자료형 bool에 대한 이해

- C++ 기본 자료형
- 참을 의미하는 `true`, 거짓을 의미하는 `false` 중 하나의 값

❖ `true` & `false`

- 1과 0이 아니라, 논리적인 참과 거짓을 의미
- `int`형 데이터로 형 변환 시 1과 0이 됨

Bool형 예



```
#include <stdio.h>
const int TRUE=1;
const int FALSE=0;
int IsPositive(int i)
{
    if (i<0)
        return FALSE;
    else
        return TRUE;
}
int main(void)
{
    int num;
    int result;

    printf("숫자 입력 : ");
    scanf("%d", &num);
    result=IsPositive(num);
    if (result==TRUE)
        printf("Positive number %dn");
    else
        printf("Negative number %dn");
    return 0;
}
```

```
#include <iostream>
using std::cin;
using std::cout;
using std::endl;
bool IsPositive(int i)
{
    if(i<0)
        return false;
    else
        return true;
}

int main(void)
{
    int num;
    bool result;

    cout<<"숫자 입력 : ";
    cin>>num;
    result=IsPositive(num);
    if (result==true)
        printf("Positive number %dn");
    else
        printf("Negative number %dn");
    return 0;
}
```



레퍼런스 (reference)

Jong Hyuk Park

레퍼런스



- ❖ 레퍼런스는 한 변수를 여러 개의 다른 이름을 사용하여 참조하는 것으로 이미 정의된 변수를 다른 이름의 레퍼런스로 사용

자료형 &레퍼런스 = 변수 ;

```
int &ref = val;
```

레퍼런스



```
int main(void)
{
    int val=10;

    int *pVal=&val; // 주소값을 얻기위해 &연산자 사용
    int &rVal=val;  //레퍼런스 선언을 위해 &연산자 사용

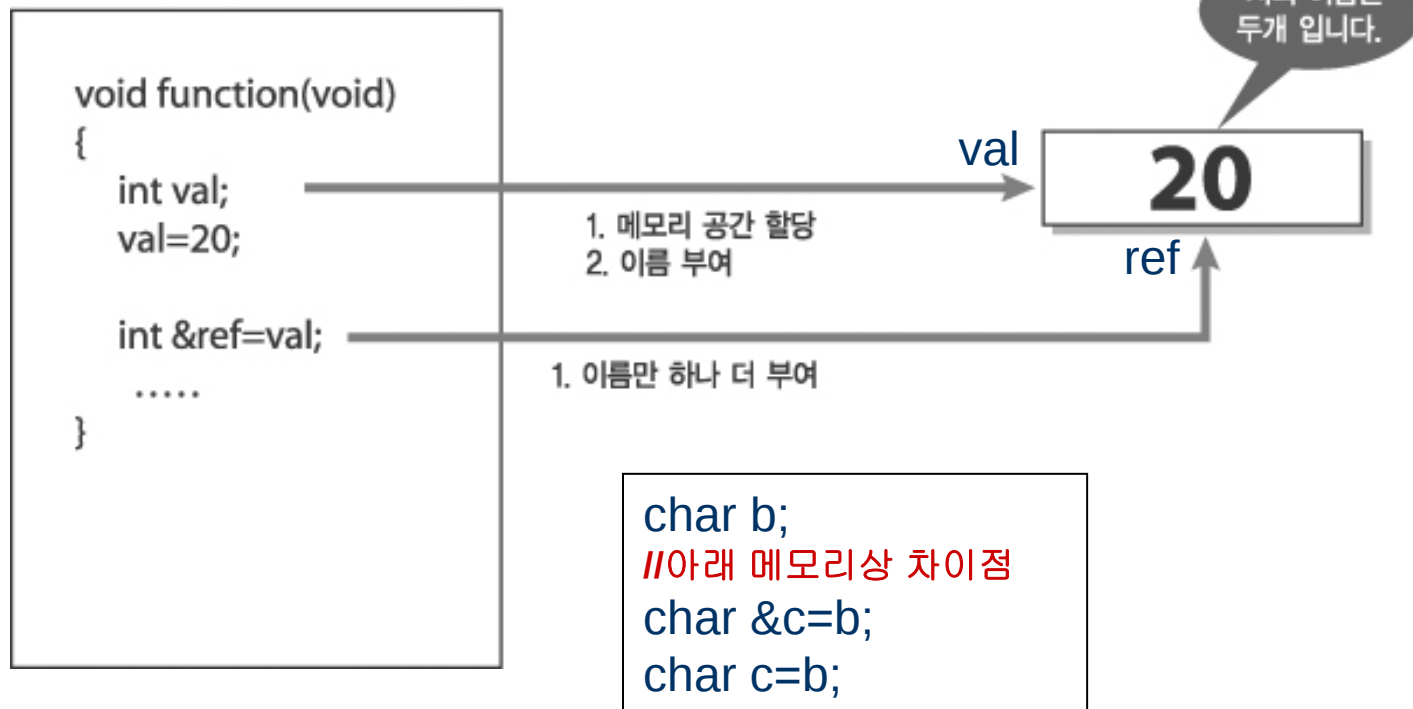
    return 0;
}
```

레퍼런스 이해



❖ 레퍼런스에 대한 또 다른 접근

- 변수란? 메모리 공간에 붙여진 이름!
- 메모리 공간에 이름을 하나 더 추가하는 행위



레퍼런스 이해



❖ 레퍼런스와 변수(변수의 이름)

- 생성되는 방법에 있어서만 차이를 보임
- 만들어지고 나면 완전히 같은 것!

```
int function(void)
{
    int val;
    val=20;
    int &ref=val;
    return val;
}
```

```
int function(void)
{
    int val;
    val=20;
    int &ref=val;
    return ref;
}
```

그림 2-3

```
void function(void)
{
    int val;
    val=20;
    int &ref1=val;
    int &ref2=ref1;
}
```

```
void function(void)
{
    int val;
    val=20;
    int &ref1=val;
    int &ref2=val;
}
```

그림 2-4

레퍼런스 예



```
#include <iostream>
using std::cout;
int main(void)
{
    int a, b = 10;
    int &c = a, &d = b;    // c,d는 변수 a,b의 참조자
    a = 5;

    cout << "a: " << a << "Wtb : " << b << 'Wn';
    cout << "c: " << c << "Wtd : " << d << "WnWn";
    ++a; --b;
    cout << "a: " << a << "Wtb : " << b << 'Wn';
    cout << "c: " << c << "Wtd : " << d << "WnWn";
    ++c; --d;
    cout << "a: " << a << "Wtb : " << b << 'Wn';
    cout << "c: " << c << "Wtd : " << d << 'Wn';
    return 0;
}
```

a: 5	b : 10
c: 5	d : 10

a: 6	b : 9
c: 6	d : 9

a: 7	b : 8
c: 7	d : 8

레퍼런스 이해



❖ 레퍼런스의 제약

- 이름이 존재하지 않는 대상을 레퍼런스 할 수 없다.
선언과 동시에 반드시 초기화되어야 한다.

```
int main(void)
{
    int &ref1;           // 초기화되지 않았으므로 ERROR!
    int &ref2=10;         // 상수가 올 수 없으므로 ERROR!

    int val=10;
    ref1=val;
    . . . . .
```

레퍼런스 인수 전달



❖ 함수의 정의 시 가인수를 레퍼런스로 정의하면 주소에 의한 인수전달(call by reference)

- 함수 내에서 가인수인 레퍼런스의 변경은 함수 호출 시의 인수인 실인수를 변경
- 기존 C 언어의 포인터를 이용하여 인수를 전달과 동일하나 함수 내에서 * 연산자를 사용하여 포인터 변수임을 명시해야 하는 번거로움 제거

```
int a;  
func(&a);  
.....  
void func(int *x)  
{  
    *x =  
    .....  
}
```

```
int a;  
func(a);  
.....  
void func(int &x)  
{  
    x =  
    .....  
}
```

레퍼런스 인수 예 1



```
// 두 수의 a 교환
#include <stdio.h>

void swap(int *a, int *b)
{
    int temp=*a;
    *a = *b;
    *b = temp;
}

int main(void)
{
    int val1=10;
    int val2=20;

    swap(&val1,&val2);
    printf(" val1: %d\n", val1);
    printf(" val2: %d\n", val2);
    return 0;
}
```

```
// 두 수의 교환
#include <iostream>
using std::cout;
using std::endl;

void swap(int &a, int &b)
{
    int temp=a;
    a = b;
    b = temp;
}

int main(void)
{
    int val1=10;
    int val2=20;

    swap(val1,val2);
    cout << " val1: " << val1 << endl;
    cout << " val2: " << val2 << endl;

    return 0;
}
```

레퍼런스 인수 전달



❖ 포인터를 이용한 Call-By-Reference

- 함수 외부에 선언된 변수의 접근이 가능
- 포인터 연산에 의해서 가능한 것이다!
- 따라서 포인터 연산의 위험성 존재!
- swap1.cpp

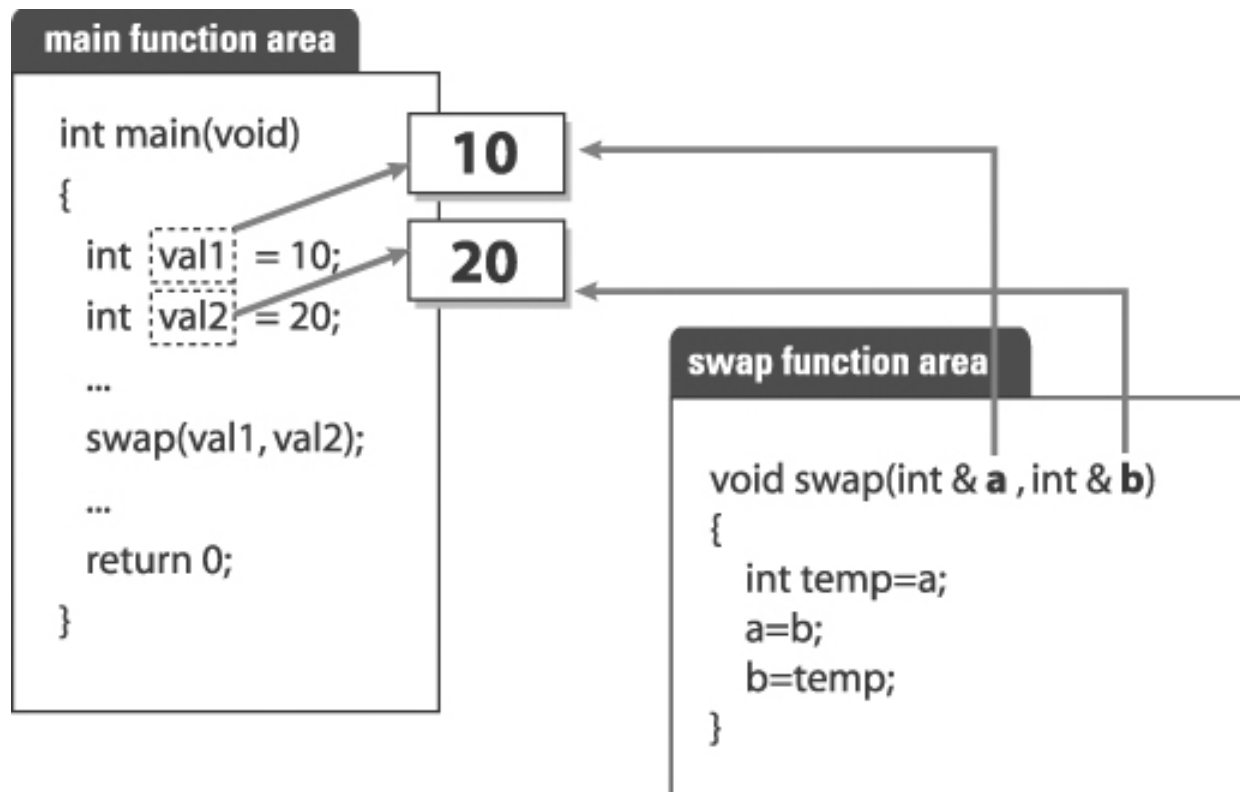
```
void swap(int *a, int *b)
{
    int temp=*a;
    a++;           //잘못 들어간 코드
    *a=*b;
    *b=temp;
}
```

레퍼런스 인수 예 1 이해



❖ 레퍼런스를 이용한 Call-By-Reference

- 함수 외부에 선언된 변수의 접근이 가능
- 포인터 연산을 사용하지 않음 → 보다 안정적임.
- 함수호출형태의 구분이 어려움.



레퍼런스를 이용한 성능의 향상



❖ 부담스러운 Call-By-Value

- 함수 호출 시 인자 전달 과정에서 발생
- 데이터를 복사하는 과정에서 발생
- P75, 예제확인 → 44바이트 메모리크기 복사

❖ 대신할 수 있는 Call-By-Reference

- 전달되는 인자를 레퍼런스로 받으면, 데이터의 복사 연산이 필요 없다.
- 원본 데이터의 손실 예상! const 선언!
- reffunc.cpp

레퍼런스 인수 예 2



```
#include <iostream>
using std::cout;
using std::endl;
using std::cin;

struct _Person {
    int age;          // 나이
    char name[20];    // 이름
    char personalID[20]; // 주민등록 번호.
};

typedef struct _Person Person;

void ShowData(Person &p)
{
    //→ void ShowData(const Person &p)
    cout<<"***** 개인 정보 출력 *****"<<endl;
    cout<<"이   름 : "<<p.name<<endl;
    cout<<"주민번호 : "<<p.personalID<<endl;
    cout<<"나   이 : "<<p.age<<endl;
}
```

```
int main(void)
{
    Person man;

    cout<<"이   름 : ";
    cin>>man.name;

    cout<<"나   이 : ";
    cin>>man.age;

    cout<<"주민번호 : ";
    cin>>man.personalID;

    ShowData(man);
    return 0;
}
```

레퍼런스를 이용한 성능의 향상 (정리)



Call by value

```
void ShowData(Person p)
{
    cout<<"***** 개인 정보 출력 *****"<<endl;
    ..... 생      략 .....
}
```



Call by reference

```
void ShowData(Person& p)
{
    cout<<"***** 개인 정보 출력 *****"<<endl;
    ..... 생      략 .....
}
```



```
void ShowData(const Person& p)
{
    cout<<"***** 개인 정보 출력 *****"<<endl;
    ..... 생      략 .....
}
```

Call by reference 에서 문제발생을 고려
→ 레퍼런스를 상수화함.
“레퍼런스를 통한 데이터 조작 허용을 불허”

레퍼런스의 반환



❖ 레퍼런스는 **함수의 반환형**으로도 선언되어 사용

- 주소가 반환
- 함수 호출 시 함수는 대입 연산자(=)의 좌우측 어느 자리에도 올 수 있음

```
#include <iostream>
using std::cout;

int &func(void); // 정수형의 레퍼런스를 반환하는 함수
int x = 88;
int main(void)
{
    func() = 100;
    // 함수 func()의 반환 레퍼런스(x)에 100 대입
    cout << "x : " << x << "\n";

    return 0;
}
int &func(void)
{ return x; }
```

x : 100

함수 func()는 x의 레퍼런스를 반환하므로 따라서 함수 func() = 100은 x = 100과 같은 의미를 갖는다.

C++ Programming

레퍼런스의 반환



```
/* ref_return.cpp */
```

```
int& increment(int &val)
{
    val++;
    return val;
}

int main(void)
{
    int n=10;
    int &ref=increment(n);

    cout<<"n : "<<n<<endl;
    cout<<"ref: "<<ref<<endl;
    return 0;
}
```

```
/* ref_error.cpp */
```

```
int& function(void)
{
    int val=10;
    return val;
}

int main(void)
{
    int &ref=function();
    cout<<ref<<endl;
    return 0;
}
```



new & delete

Jong Hyuk Park

동적할당,해제



- ❖ 프로그램의 시작에서는 필요한 메모리의 크기를 모르고 프로그램이 실행되는 동안에 그 크기가 결정되거나 메모리를 효율적으로 사용하기 위해서는 프로그램 실행 시에 메모리를 **동적할당(dynamic allocation)**

- ❖ C 언어
void *malloc(자료형 크기);
void *calloc(자료형 개수,자료형 크기);
- ❖ C++ 언어
new 자료형;
new 자료형[자료형 개수];

- ❖ C 언어
void free(변수);
- ❖ C++ 언어
delete 변수;
delete []배열변수;

동적할당,해제



❖ new와 delete 연산자의 기본적인 기능

- malloc_free.cpp, new_delete.cpp

```
int main(void)
{
    int * val = new int;

    int * arr = new int[size];
    . . . . .

    delete val;

    delete []arr;
    . . . . .
```

```
int main(void)
{
    int * val = (int*)malloc(sizeof(int));

    int * arr = (int*)malloc(sizeof(int)* size);
    . . . . .

    free(val);

    free(arr);
    . . . . .
```

동적할당,해제예



❖ long int 형 (4 바이트) 동적할당, 해제

- ❖ C 언어
`long *p = malloc(sizeof(long));`
.....
`free(p);`
- ❖ C++ 언어
`long *p = new long;`
.....
`delete p;`

❖ long int 형 10개 (40 바이트) 동적할당, 해제

- ❖ C 언어
`long *p = calloc(10,sizeof(long));`
.....
`free(p);`
- ❖ C++ 언어
`long *p = new long[10];`
.....
`delete []p;`

동적할당,해제예



❖ NULL 포인터 리턴하는 new 연산자

- 메모리 할당 실패 시 NULL 포인터 리턴!
- 새로운 표준에 관한 내용은 "예외 처리"를 통해서 다시 언급 (13장)!
- NULL_new.cpp

```
int* arr=new int[size];           // 배열의 동적 할당
if(arr==NULL)                     // 동적 할당 검사
{
    cout<<"메모리 할당 실패"<<endl;
    return -1;                    //프로그램 종료
}
```

- Debug_new.cpp

new & delete 예



```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    int size, *arr, i, j;
    printf("할당하고자 하는 배열의 크기: ");
    scanf("%d", &size);
    arr=(int*)malloc(sizeof(int)*size);
        // 배열의 동적 할당.
    if (arr==NULL) {
        printf("메모리 할당 실패\n");
        return -1; //프로그램 종료.
    }
    for(i=0; i<size; i++)
        arr[i]=i+10;
    for(j=0; j<size; j++)
        printf("arr[%d]=%d\n",j,arr[j]);
    free(arr); // 할당된 메모리 소멸.
    return 0;
}
```

```
#include <iostream>
using std::cin;
using std::cout;
using std::endl;
int main(void)
{
    int size;
    cout<<"할당하고자 하는 배열의 크기: ";
    cin>>size;
    int* arr=new int[size]; // 배열의 동적 할당.
    if(arr==NULL)
    {
        cout<<"메모리 할당 실패"<<endl;
        return -1; //프로그램 종료.
    }
    for(int i=0; i<size; i++)
        arr[i]=i+10;
    for(int j=0; j<size; j++)
        cout<<"arr["<<j<<"]="<<arr[j]<<endl;
    delete []arr; // 할당된 메모리 소멸.
    return 0;
}
```

HW 및 실습



1. call by value 와 call by reference (2가지형태) 설명 후, 예제를 활용하여 실제 코딩후 실행결과를 출력하라 (HW #01).
2. 1장 과제의 은행계좌 관리 프로그램에 다음을 수정하여라 (실습).
 - ❖ Account pArray[100];을 Account *pArray; 로 수정
 - 프로그램 시작 시 메모리를 동적할당으로 초기화
 - 프로그램 종료 시에 할당된 메모리를 해제



Q & A

Jong Hyuk Park