2장. C 기반의 C++ 2

담당교수 : 김미경

수업 내용

- 메모리 구조와 동적 할당
  - stack의 용도 및 특징
  - heap의 용도 및 특징
  - malloc & free 함수의 필요성
- 자료형 bool
- reference 의 이해
- 레퍼런스와 함수
- new & delete

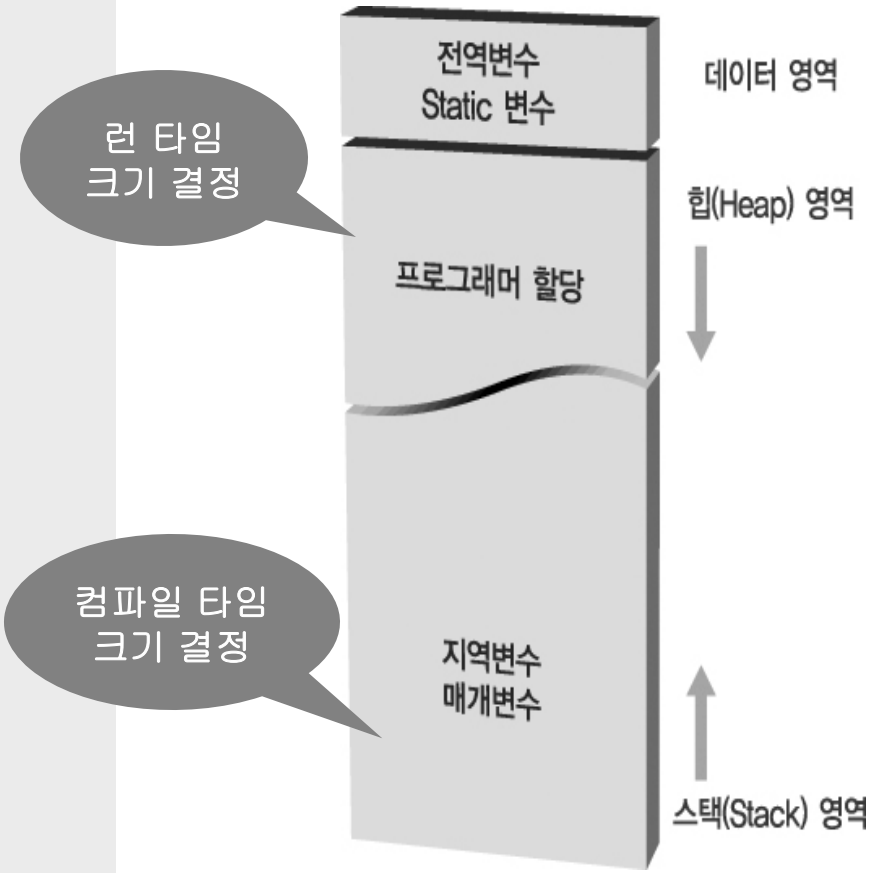
2-1 메모리 구조와 동작 할당

문제점?

```
void function (int);

int main(void)
{
    int size;
    cin>>size;
    function(size);
    return 0;
}

void function(int i)
{
    int array[i];
}
```



2-1 메모리 구조와 동작 할당

```c
/* memory.c */
#include <stdio.h>
void fct1(int);
void fct2(int);

int a=10;
int b=20;

int main()
{
    int m=123;

    fct1(m);
    fct2(m);
    return 0;
}
void fct1(int)
{
    int d=30;
}
void fct2(int){
    int f=40;
}
```

C++11 이후 배열 칸 변수 가능

```c
/* ProbArray.c */

#include <stdio.h>

void function(int);

int main(void)
{
   int m = 0;
   scanf("%d", &m);    // 배열의 길이 입력 받음
   function(m);        // 배열의 길이 전달

  return 0;
}

void function(int i)
{
    int array[i];    // 입력받은 길이만큼 배열 할당
}
```

C++ 표준

연도	C++표준	비공식 명칭
1998년	ISO/IEC 14482: 1998	C++98
2003년	ISO/IEC 14482: 2003	C++03
2007년	ISO/IEC TR 19768:2007	C++07/TR1
2011년	ISO/IEC 14482: 2011	C++11 ➔기능의 많은 변화
2014년	ISO/IEC 14482: 2014	C++14
2017년	ISO/IEC 14482: 2017	C++17

C언어 표준

연도	명칭	
1989년	C89	
1990년	C90	ISO표준, ANSI에서도 받아들임, C89와 동일, 약간 에러 수정
1995년	C95	
1999년	C99	For loop 에 변수 초기화 변수 선언 가능
2011년	C11	

Heap 영역의 필요성

- 배열 선언시 반드시 상수만 사용해야 하는 이유는?
  - 스택과 데이터 영역에 할당된 메모리의 크기는 컴파일 되는 동안(compile-time)에 결정되어야 한다.

- 할당해야 할 메모리의 크기를 런-타임에 결정해야 하는 경우, 유용하게 사용되는 메모리 공간이 바로 힙 영역이다.

Heap 영역의 필요성

- **배열의 선언**
  - **배열의 길이 선언은 상수!**
  - **컴파일 타임에 요구되는 메모리 공간의 크기를 결정해야 함**

- 예제 ProbArray2.c에서 function 함수는 에러가 있는가 ?

- 문제가 있다면 이유는?

  -> 지역변수 i 가 10으로 초기화 되는 순간은 컴파일되는 동안(compile-time)이 아니라, 실행되는 동안(run-time)에 결정됨.

◆할당해야 할 메모리의 크기를 런-타임(프로그램이 실행되는 동안)에 결정해야 하는 경우, 유용하게 사용되는 메모리 공간이 힙(Heap)영역이다

```
/* ProbArray2.c? */

#include <stdio.h>

void function();

int main(void)
{
  int m = 0;
  function(); // 배열의 길이 전달

  return 0;
}

void function()
{
   int i = 10;   // 4 바이트
   int array[i]; //  40 바이트 예상
}
```

- **malloc 함수의 활용**

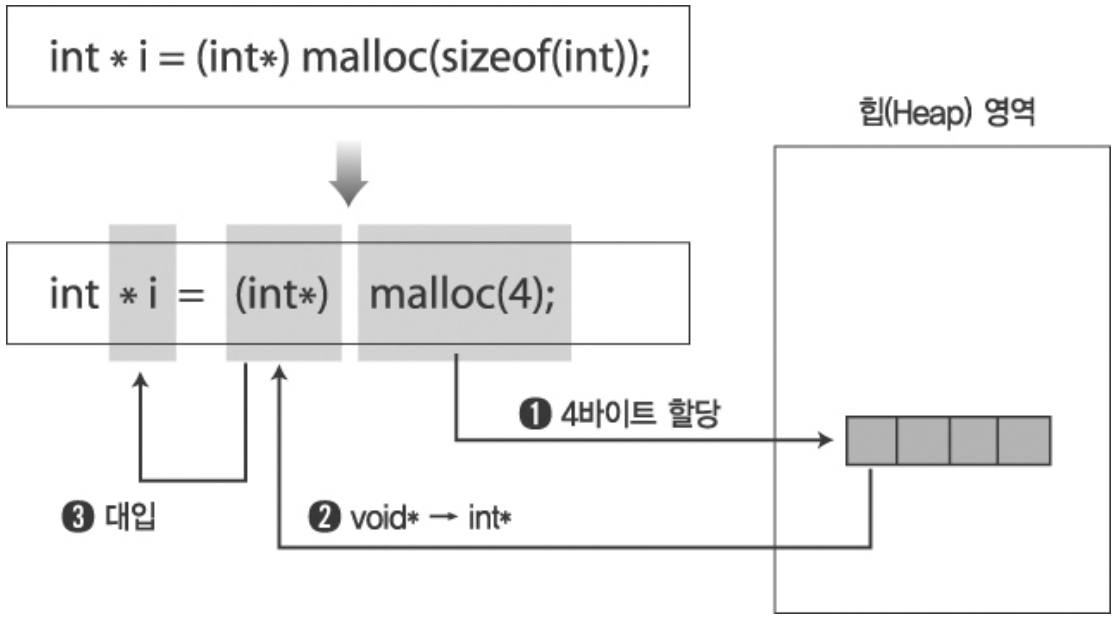


그림 2-8: malloc 함수의 호출

C에서의 동적 할당

- 힙은 프로그래머가 관리하는 메모리 공간이므로 메모리를 해제하는 것도 프로그래머가 신경 써야 할 내용

- 동적 할당된 메모리 공간의 소멸(해제) 필요 – free 함수 이용

```
1:  /* malloc&free.c */


2:  #include <stdio.h>
3:  #include <stdlib.h>
4:
5:  int main (void)
6:  {
7:     int* a;
8:     a=(int*)malloc(sizeof(int)); // 메모리 할당.
9:     if(a==NULL)   // 메모리 할당의 성공 유무 확인
10:    {
11:          puts("메모리 할당에 실패!");
12:          exit(1);
13:    }
14:
15:    *a=20;
16:    printf("힙에 저장된 변수 a : %d ₩n", *a);
17:    free(a); // 메모리 해제.
18:    return 0;
19:  }
```

C에서의 동적 할당

```
/* ProbArray2.c */

#include <stdio.h>
#include <stdlib.h>

void function(int);
int main (void)
{
        int m=0;
        fputs("배열의 크기를 입력하세요 : ", stdout);
        scanf("%d", &m);
        function(m);

        return 0;
}

void function(int i)
{
        //int array[i];   // ProbArray.c에서의 문제점.

        int* array =(int*)malloc(sizeof(int)*i);   //동적 메모리 할당.
        int j;
        if(array==NULL)
        {
                puts("메모리 할당에 실패!");
                exit(1);
        }
        /* 동적 할당한 메모리 사용 */
        for(j=0; j<i; j++)
                array[j]=j+1;
        for(j=0; j<i; j++)
                printf("%d ", array[j]);
        printf("\n");

        free(array);      // 할당된 메모리 소멸.
}
```

2-2 C++동적 할당 – new & delete

- **new와 delete 연산자의 기본적 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);
    . . . . .
```

2-2 C++동적 할당 – new & delete

```
2      malloc_free.cpp
3  */
4
5  #include <iostream>
6  #include <stdlib.h>
7  using std::cin;
8  using std::cout;
9  using std::endl;
10
11 int main(void)
12 {
13     int size;
14     cout<<"할당하고자 하는 배열의 크기: ";
15     cin>>size;
16     // 배열의 동적 할당.
17     int* arr=(int*)malloc(sizeof(int)*size);
18
19     for(int i=0; i<size; i++)
20         arr[i]=i+10;
21
22     for(int j=0; j<size; j++)
23         cout<<"arr["<<j<<"]= "<<arr[j]<<endl;
24
25     free(arr);  // 할당된 메모리 소멸.
26
27     return 0;
28 }
```

```
1  /*   new_delete.cpp*/
2
3  #include <iostream>
4
5  using std::cin;
6  using std::cout;
7  using std::endl;
8
9  int main(void)
10 {
11     int size;
12     cout<<"할당하고자 하는 배열의 크기: ";
13     cin>>size;
14
15     int* arr=new int[size];  // 배열의 동적 할당.
16
17     for(int i=0; i<size; i++)
18         arr[i]=i+10;
19
20     for(int j=0; j<size; j++)
21         cout<<"arr["<<j<<"]= "<<arr[j]<<endl;
22
23     delete []arr;  // 할당된 메모리 소멸.
24
25     return 0;
26 }
```

2-2 C++동적 할당 - new & delete

- **NULL 포인터 리턴하는 new 연산자**
  - **메모리 할당 실패 시 NULL 포인터 리턴!**
  - **새로운 표준에 관한 내용은 "예외 처리"를 통해서 다시 언급!**
  - **NULL_new.cpp, Debug_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
}
```

2-2 C++동적 할당 – new & delete

```cpp
/*   NULL_new.cpp*/
#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;
}
```

```cpp
/*   Debug_new.cpp*/
#include <iostream>
//#define DEBUG 1;
#define DEBUG  0;

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 DEBUG==1
    cout<<"디버그 모드 입니다"<<endl;
    if(arr==NULL)
    {
        cout<<"메모리 할당 실패"<<endl;
        return -1; //프로그램 종료.
    }
#endif

    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;
}
```

틀린데 찾아 보기

2-3 자료형 bool

- 자료형 bool에 대한 이해
  - 기본 자료형의 일종
  - 참을 의미하는 true, 거짓을 의미하는 false 중 하나의 값을 지닌다.

- true & false
  - 1과 0이 아니라, 논리적인 참과 거짓을 의미하는 키워드이다.
  - int형 데이터로 형 변환 시 1과 0이 된다.
  - bool1.cpp, bool2.cpp, bool3.cpp

2-3 자료형 bool

```c
/*  bool1.cpp
    C 스타일의 프로그램.*/
#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 \n");
    else
        printf("Negative number \n");

    return 0;
}
```

```c
/*  bool2.cpp
    C++ 스타일의 프로
#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 \n");
    else
        printf("Negative number \n");

    return 0;
}
```

```c
/*  bool3.cpp
    true & false*/

#include <iostream>

using std::cout;
using std::endl;

int main(void)
{
    int BOOL=true;
    cout<<"BOOL : "<<BOOL<<endl;

    BOOL=false;
    cout<<"BOOL : "<<BOOL<<endl;

    return 0;
}
```

2-4 레퍼런스의 이해

- **레퍼런스의 개념**
  - **이름 지니는 대상에 별명을 붙여주는 행위**
  - **reference.cpp**

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

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

    int *pVal=&val;        // 주소 값을 얻기 위해 & 연산자 사용
    int &rVal=val;         // 레퍼런스 선언을 위해 & 연산자 사용

    return 0;
}
```

```
1  /*  reference.cpp*/
2
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  int main(void)
9  {
10     int val=10;
11     int &ref=val;
12
13     val++;
14     cout<<"ref : "<<ref<<endl;
15     cout<<"val : "<<val<<endl;
16
17     ref++;
18     cout<<"ref : "<<ref<<endl;
19     cout<<"val : "<<val<<endl;
20
21     return 0;
22 }
```

2-4 레퍼런스의 이해

- 레퍼런스에 대한 또 다른 접근
  - 변수란? 메모리 공간에 붙여진 이름!
  - 메모리 공간에 이름을 하나 더 추가하는 행위

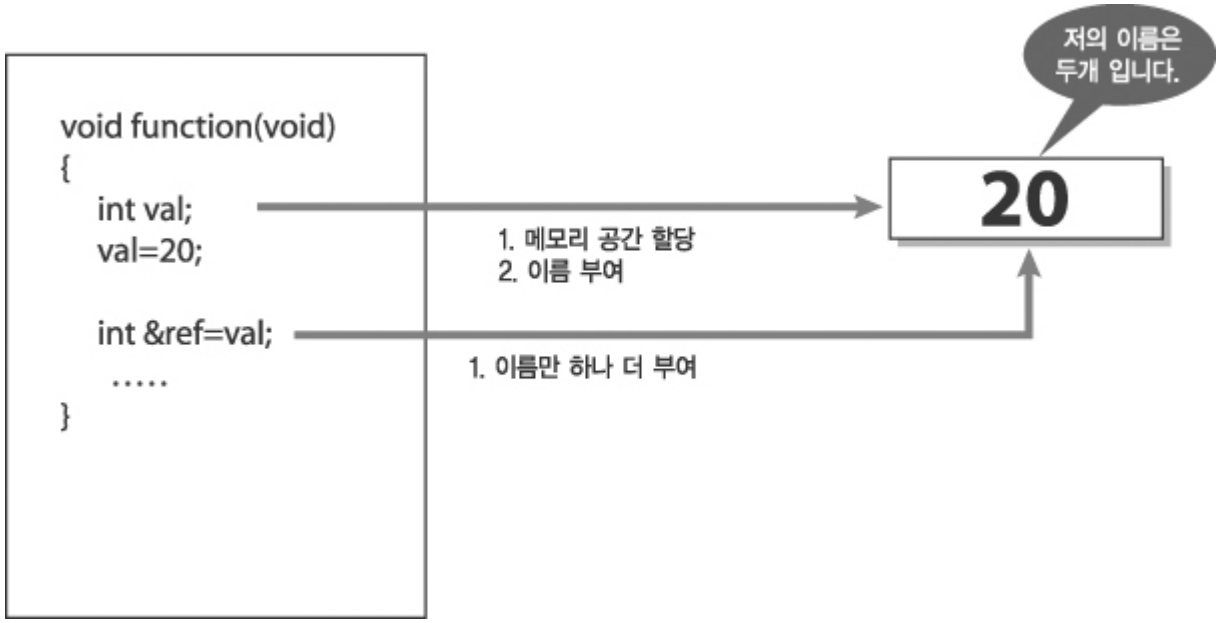


그림 2-2

2-4 레퍼런스의 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

2-4 레퍼런스의 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;
    . . . . .
```

2-5 레퍼런스와 함수

- **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;
}
```

```
1  /*  swap1.cpp*/
2
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  void swap(int *a, int *b)
9  {
10     int temp=*a;
11     *a=*b;
12     *b=temp;
13 }
14
15 int main(void)
16 {
17     int val1=10;
18     int val2=20;
19
20     cout<<"val1:"<<val1<<' ';
21     cout<<"val2:"<<val2<<endl;
22
23     swap(&val1, &val2);
24     cout<<"val1:"<<val1<<' ';
25     cout<<"val2:"<<val2<<endl;
26
27     return 0;
28 }
```

2-5 레퍼런스와 함수

- **레퍼런스를 이용한 Call-By-Reference**
  - 함수 외부에 선언된 변수의 접근이 가능
  - 포인터 연산을 할 필요가 없으므로 보다 안정적이다
  - 함수의 호출 형태를 구분하기 어렵다!
  - swap2.cpp (p128)

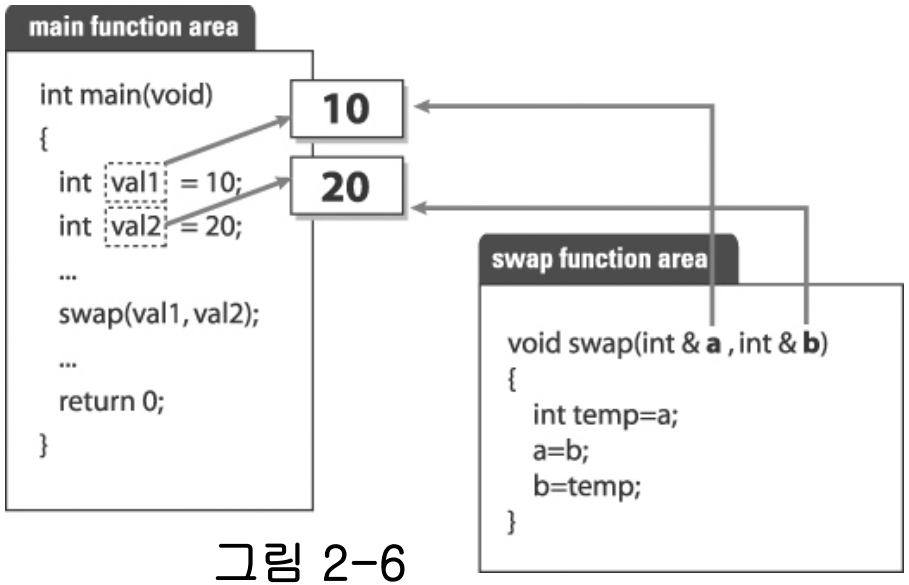


그림 2-6

```
1  /*  swap2.cpp*/
2  #include <iostream>
3  using std::cout;
4  using std::endl;
5  void swap(int &a, int &b)
6  {
7      int temp=a;
8      a=b;
9      b=temp;
10 }
11 int main(void)
12 {
13     int val1=10;
14     int val2=20;
15
16     cout<<"val1:"<<val1<<' ';
17     cout<<"val2:"<<val2<<endl;
18
19     swap(val1, val2);
20     cout<<"val1:"<<val1<<' ';
21     cout<<"val2:"<<val2<<endl;
22
23     return 0;
24 }
```

2-5 레퍼런스와 함수

- 부담스러운 Call-By-Value
  - 함수 호출 시 인자 전달 과정에서 발생
  - 데이터를 복사하는 과정에서 발생

- 대신할 수 있는 Call-By-Reference
  - 전달되는 인자를 레퍼런스로 받으면, 데이터의 복사 연산이 필요 없다.
  - 원본 데이터의 손실 예상! const 선언!
  - reffunc.cpp

2-5 레퍼런스와 함수

```
1  /*  reffunc.cpp*/
2  #include <iostream>
3  using std::cout;using std::endl;using std::cin;
4  struct _Person {
5      int age;          // 나이
6      char name[20];    // 이름
7      char personalID[20]; // 주민등록 번호.
8  };
9  typedef struct _Person Person;
10 void ShowData(const Person &p)
11 {
12     cout<<"******* 개인 정보 출력 *******"<<endl;
13     cout<<"이      름 : "<<p.name<<endl;
14     cout<<"주 민 번 호 : "<<p.personalID<<endl;
15     cout<<"나      이 : "<<p.age<<endl;
16 }
17 int main(void){
18     Person man;
19     cout<<"이      름 : ";
20     cin>>man.name;
21     cout<<"나      이 : ";
22     cin>>man.age;
23     cout<<"주 민 번 호 : ";
24     cin>>man.personalID;
25     ShowData(man);
26     return 0;
27 }
```

2-6 레퍼런스를 이용한 성능의 향상

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

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

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

2-7 레퍼런스를 리턴하는 함수의 정의

- **예제의 분석, 이해 그리고 응용!**

```
/*  ref_return.cpp  */

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

int main(void)
{
    int n=10, &ref;
    cout<<"n  : "<<n<<endl;
    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;
 }
```

2-8 간단한 파일 입출력

- #include <fstream>
- using namespace std;
- ifstream : 파일을 읽을때
- ofstream : 파일에 쓸때
- cout, cin과 비슷

2-8 간단한 파일 입출력

파일 열기와 닫기

1. ifstream fin; //개체 선언
2. fin.open("test.txt"); //파일 열기

    1+2. ifstream fin("test.txt"); //개체선언과 열기

3. fin.close(); //파일 닫기

2-8 간단한 파일 입출력

- 간단한 파일 입출력 파일 읽기
  - test.txt

    홍길동

    200512345

```
1  /* file.read1.cpp */
2  #include <iostream>
3  #include <fstream>
4  using namespace std;
5
6  int  main()
7  {
8       char name[10];
9       unsigned student_id;
10      ifstream fin;
11      fin.open("test.txt");
12      fin>> name >> student_id;
13      cout << "이 름  : " << name << ", 학 번 :" << student_id << endl;
14 }
```

2-8 간단한 파일 입출력

- 파일 쓰기
  - test2.txt

?

```
1  /*& file_write1.cpp */
2  #include <iostream>
3  #include <fstream>
4  using namespace std;
5
6  void main()
7  {
8      char name[10];
9      unsigned student_id;
10     ofstream fout;
11     cout << "이 름  : " ;
12     cin >> name;
13     cout << "학 번  : ";
14     cin >> student_id;
15     fout.open("test2.txt");
16     fout <<  name << student_id <<endl;
17     fout.close();
18 }
```

2-8 간단한 파일 입출력

파일 읽기 관련 함수

- ifstream fin;
- char ch;
- char line[100];
- fin.eof();
- //파일의 끝이면 1을 리턴, 아니면 0을 리턴
- fin.get(ch);
- //파일에서 한문자를 읽는다
- fin.getline(line,100);
- //파일에서 한 행을 읽는다.

```
1  /* file_get1.cpp*/
2  #include <iostream>
3  #include <fstream>
4  using namespace std;
5  int main()
6  {
7      char ch, line[200];
8      ifstream fin("article.txt");
9      if(fin == NULL)
10     {
11         cout << "파일이 존재하지 않습니다 " << endl;
12         return 0;
13     }
14     while(fin.get(ch))
15     {
16         cout<<ch;
17     }
18     fin.close();
19     cout << "파일 읽기 종료"<<endl<<endl;
20
21 //  fin.open("article.txt",ios_base::in);
22     fin.open("article.txt");
23     while(fin.getline(line, 200))
24     {
25         cout<<line<<endl;
26     }
27     fin.close();
28 }
```

2-8 간단한 파일 입출력

- 파일 모드(1/2)
  ifstream fin("test.txt",mode1 | mode2);
- mode
  - ios_base::in 파일을 읽기 위해 연다
  - ios_base::out 파일을 쓰기 위해 연다
  - ios_base::ate 파일을 열 때 파일 끝을 찾는다
  - ios_base::app 파일 끝에 덧붙인다
  - ios_base::trunc 파일이 이미 존재하면 파일 내용을 지운다
  - ios_base::binary 2진파일로 연다

2-8 간단한 파일 입출력

- 파일 모드
  - 밑에서 3번째줄에 모드가 있을 경우와 없을 경우 test2.txt파일의 출력 결과는?
  - 모드가 있을경우
    - ios_base::app와 ios_base::trunc의 차이는?

```
1  /*& file_write2.cpp */
2  #include <iostream>
3  #include <fstream>
4  using namespace std;
5
6  int  main()
7  {
8      char name[10];
9      unsigned student_id;
10     ofstream fout;
11     cout << "이 름  : " ;
12     cin >> name;
13     cout << "학 번  : ";
14     cin >> student_id;
15     fout.open("test2.txt");
16     fout <<  name << student_id <<endl;;
17     fout.close();
18
19     fout.open("test2.txt", ios_base::out | ios_base::app);
20     fout << "Hello World" << endl;
21     fout.close();
22 }
```

2-8 간단한 파일 입출력

- 출력 형태 지정
  - width, fill, precision
    - ex) cout.width(10); fout.width(10)
    - cout.fill('*'), cout.fill(' ')
    - cout.precision(2), fout.precision(2) : 전체 표현 자리수
  - <iomanip> include 할 경우
  - setw(10)

```
1  /*& file_write3.cpp */
2  #include <iostream>
3  #include <fstream>
4  #include <iomanip>
5  using namespace std;
6
7  int  main()
8  {
9      char name[10];
10     unsigned student_id;
11     ofstream fout;
12     cout << "이름 : " ;
13     cin >> name;
14     cout << "학번 : ";
15     cin >> student_id;
16     fout.open("test2.txt");
17     fout.width(10);
18     fout <<  name;
19     fout.width(20);
20     fout << student_id;
21     fout<<endl;;
22     fout.close();
23
24     fout.open("test2.txt", ios_base::out | ios_base::app);
25     fout.fill('1');
26     fout << setw(20) << "Hello World" << endl;
27     fout.close();
28 }
```

assignment#1

- 레포트1 제출 게시판에 비밀글로 등록
- 실행파일, 이름 : 이름_학번_Report01.exe,
- 소스파일이름:report01.cpp,
- 기한 : 다음 수업 하루 전 밤 12시

- 1. 입력파일 : basic.inp , 출력파일 : basic.out
- 2. 입력파일을 읽어서 연산기호와 숫자를 읽어서 연산을 수행하고 결과를 출력파일에 쓴다.
- 3. 한번 연산할 때 두 개의 숫자만 사용한다.
- 4. 사칙연산(*,+,-,/)만 사용한다.
- 예시
  - 입력파일 : basic.inp

    2
    +
    2 3
    *
    3 3

    연산 개수
    연산기호
    숫자 숫자

  출력파일 : basic.out

  5
  9