

함수와 기억클래스

- 기억클래스와 변수



● 기억 클래스의 종류

- 세분화된 데이터 영역에서 변수가 어디에 어떻게 확보될지 결정

구분	유효 범위	생존 기간	메모리	초기화 여부
auto(자동변수)	블록 내	일시적	스택	쓰레기값
extern(외부변수)	프로그램 내	영구적	정적자료영역	숫자 0
static(정적변수)	내부 : 블록 내 외부 : 모듈 내	영구적	정적자료영역	숫자 0
register (레지스터 변수)	블록 내	일시적	CPU 내의 레지스터	쓰레기값

스택
확장 영역
히프
정적자료 영역
텍스트 영역

기억장소의 영역

● 생존기간

- 일시적 : 변수가 선언된 시점에서 생성되었다가 블록 종료 시 자동 소멸.
프로그램 내에서 생성과 소멸이 여러 번 반복
- 영구적 : 프로그램이 시작할 때 생성되어 프로그램 실행 중에 항상 메모리상의 동일한 기억 공간에 존재. 프로그램이 완전히 끝날 때 비로소 변수 소멸

● 일반적으로 선언하는 변수는 기억클래스 생략

변수 선언 기본 형식	변수 선언 사용 예
[기억클래스] 자료형 변수명;	auto int a = 0;

● 변수 선언의 구분

- 자료형에 의한 구분 : 생략할 수 없으며 변수에 저장되는 값의 형태와 기억공간의 크기 결정 (int, char, float 등)
- 기억클래스에 의한 구분 : 생략할 수 있으며 생략할 경우 auto로 인식. 유효 범위와 생존기간 등 변수의 성격 결정 (auto, extern, static 등)

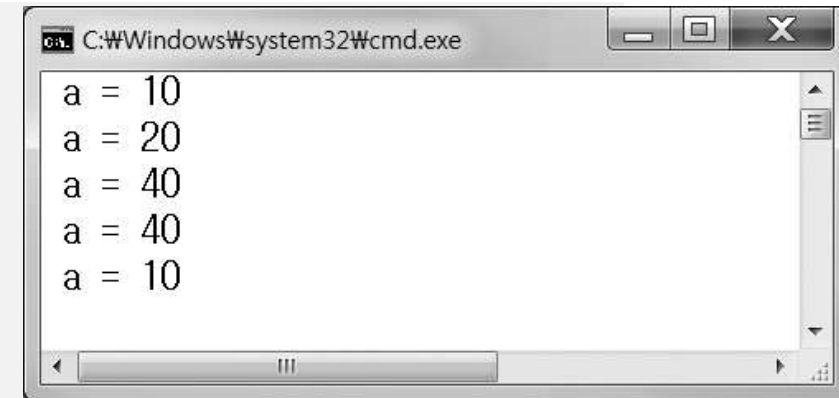
● 자동변수

- 자동변수(auto)는 함수 정의부 안에 선언된 변수
- 함수의 매개변수도 자동변수에 포함
- 프로그램이 실행되는 동안 생성과 소멸 반복
- 변수 선언 문장을 만나면 메모리가 할당되므로 변수 선언문 이후에서만 사용가능
- 변수 선언이 된 블록 밖으로 벗어나면 메모리가 해제되어 변수를 사용할 수 없음
- 일반적으로 사용하는 형태로 auto라는 기억클래스를 붙이지 않고 사용가능

● 지역변수와 전역변수

- 지역변수 : 블록(함수) 안에 선언된 변수. 사용범위가 선언된 블록 내부로 한정
- 전역변수 : 함수 외부에서 변수 선언
 - 외부변수는 대표적인 전역변수
 - 프로그램 내의 모든 함수에서 사용할 수 있는 변수
 - 프로그램이 실행되는 동안 메모리상에 항상 배치되므로 값 유지
 - 자동으로 초기화(수치 데이터라면 0으로)

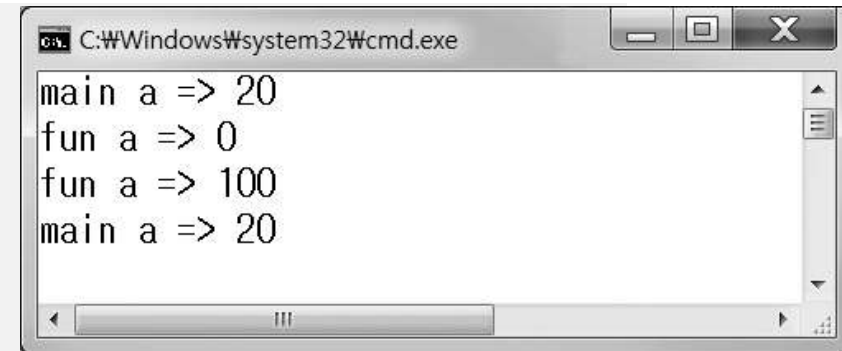
```
01 #include <iostream>
02 using namespace std;
03 void main()
04 {
05     int a = 10;
06     cout<<" a = "<<a<<endl;
07     {
08         int a = 20;
09         cout<<" a = "<<a<<endl;
10         {
11             a+=20;
12             cout<<" a = "<<a<<endl;
13         }
14         cout<<" a = "<<a<<endl;
15     }
16     cout<<" a = "<<a<<endl;
17 }
```



The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The output of the program is displayed as follows:

```
a = 10
a = 20
a = 40
a = 40
a = 10
```

```
01 #include<iostream>
02 using namespace std;
03 void fun();
04 int a; // 외부변수 a
05 void main()
06 {
07     int a = 20; // 지역변수 a
08     cout<<"\n main a => "<<a;
09     fun();
10     cout<<"\n main a => "<<a<<"\n";
11 }
12 void fun()
13 {
14     cout<<"\n fun a => "<<a;
15     a=a+100;
16     cout<<"\n fun a => "<<a;
17 }
```



```
C:\Windows\system32\cmd.exe
main a => 20
fun a => 0
fun a => 100
main a => 20
```

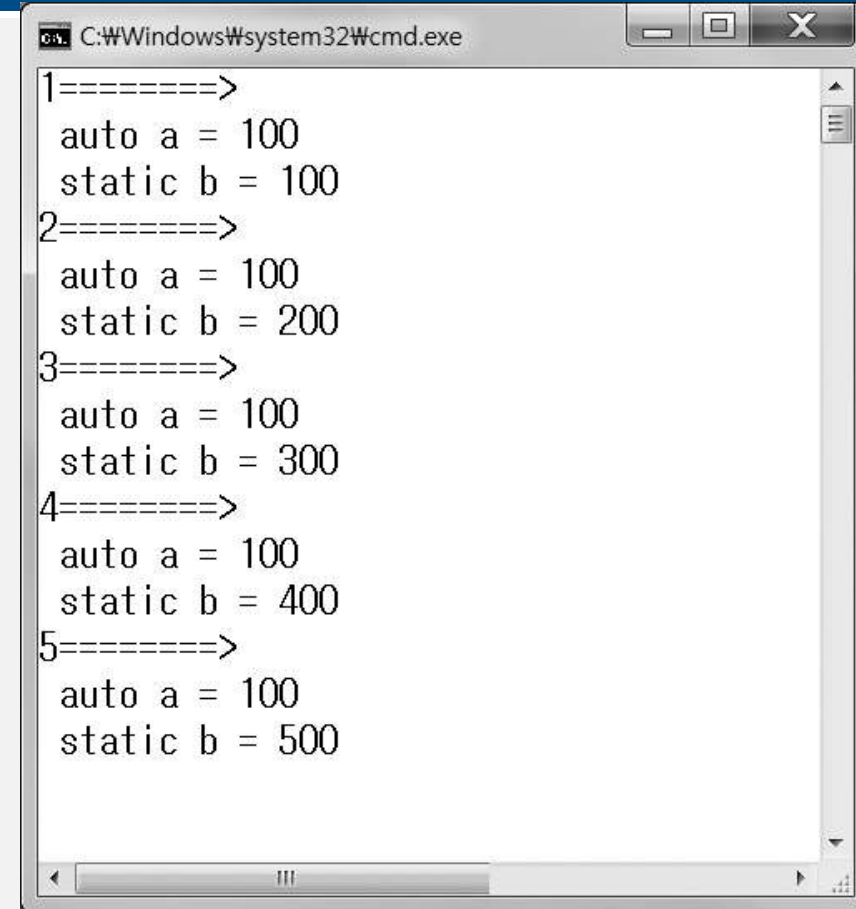
● 정적변수

- 변수를 선언할 때 변수명 앞에 static을 붙이면 정적변수로 선언
 - 지역변수로 정적변수 : 블록 내부에 변수 선언
 - 전역변수로 정적변수 : 블록 외부에 변수 선언

변수명	구분	유효 범위	생존 기간	메모리	초기화 여부	변수값 여부
auto	자동변수	블록 내	일시적	스택	쓰레기 값	유지 안됨
static	지역변수로 정적변수	블록 내	영구적	메모리	숫자 0	유지함

정적변수와 자동변수의 차이 알아보기

```
01 #include <iostream>
02 using namespace std;
03 void sub();
04 void main()
05 {
06     for(int i=1; i<=5; i++){
07         cout<< i << "=====>  n";
08         sub();
09     }
10 }
11 void sub()
12 {
13     auto int a = 0;
14     static int b = 0;
15     a+=100;
16     b+=100;
17     cout<<" auto a = "<<a<<endl;
18     cout<<" static b = "<<b<<endl;
19 }
```



```
C:\Windows\system32\cmd.exe
1=====>
auto a = 100
static b = 100
2=====>
auto a = 100
static b = 200
3=====>
auto a = 100
static b = 300
4=====>
auto a = 100
static b = 400
5=====>
auto a = 100
static b = 500
```


● 외부변수

- 전역변수로 정적변수는 그 변수가 선언되어 있는 파일에서만 사용 가능
- 프로그램을 작성할 때 파일 하나에 모든 로직을 구현할 수도 있지만 파일 여러 개로 프로그램 하나를 구성할 수 있음
→ 파일 여러 개를 모아 만든 프로그램 하나를 프로젝트라고 함
- 프로젝트로 묶여 있는 모든 파일내에서 공유해서 사용할 수 있는 변수
→ 외부변수
- 다른 파일에 선언된 외부변수에 접근하려면 사용하기 전 참조하겠다는 의미의 예약어 extern 필요

외부변수 선언 기본 형식	외부변수 선언 사용 예
extern 자료형 변수명;	extern int a;

// 1개 프로젝트에 3개 파일

// file1.cpp

```
#include<iostream>
using namespace std;
int a = 20;
int b = 30;
void sub1();
void sub2();
```

```
void main()
```

```
{
```

```
    cout << "\n main의 a (file1.cpp) ==> " << a;
```

```
    sub1();
```

```
    sub2();
```

```
    cout << "\n main의 a (file1.cpp) ==> " << a;
```

```
    cout << "\n main의 b (file1.cpp) ==> " << b << "\n";
```

```
}
```

```
// file2.cpp
```

```
#include<iostream>
```

```
using namespace std;
```

```
extern int a;
```

```
void sub1()
```

```
{
```

```
    a += 100;
```

```
    cout << "\n sub1의 a (file2.cpp) ==> " << a;
```

```
}
```

```
// file3.cpp
```

```
#include<iostream>
```

```
using namespace std;
```

```
static int b = 20;
```

```
void sub2()
```

```
{
```

```
    b += 100;
```

```
    cout << "\n sub2의 b (file3.cpp) ==> " << b;
```

```
}
```

// 1개 프로젝트에 3개 파일

// file1.cpp

```
#include<iostream>
using namespace std;
int a = 20;
int b = 30;
void sub1();
void sub2();
```

void main()

```
{
    cout << "\n main의 a (file1.cpp) ==> " << a;
    sub1();
    sub2();
    cout << "\n main의 a (file1.cpp) ==> " << a;
    cout << "\n main의 b (file1.cpp) ==> " << b << "\n";
}
```

```
// file2.cpp
#include<iostream>
using namespace std;
extern int a; // file1에 선언한 외부 변수 a사용
void sub1()
{
    a += 100;
    cout << "\n sub1의 a (file2.cpp) ==> " << a;
}
```

```
// file3.cpp
#include<iostream>
using namespace std;
static int b = 20;
void sub2()
{
    b += 100;
    cout << "\n sub2의 b (file3.cpp) ==> " << b;
}
```

```
C:\Windows\system32\cmd.exe
main의 a (file1.cpp) ==> 20
sub1의 a (file2.cpp) ==> 120
sub2의 b (file3.cpp) ==> 120
main의 a (file1.cpp) ==> 120
main의 b (file1.cpp) ==> 30
```

// 1개 프로젝트에 3개 파일

// file1.cpp

```
#include<iostream>
using namespace std;
int a = 20;
int b = 30;
void sub1();
void sub2();
```

```
void main()
{
```

```
    cout << "\n main의 a (file1.cpp) ==> " << a;
    sub1();
    sub2();
    cout << "\n main의 a (file1.cpp) ==> " << a;
    cout << "\n main의 b (file1.cpp) ==> " << b << "\n";
}
```

```
// file2.cpp
#include<iostream>
using namespace std;
extern int a; // file1에 선언한 외부 변수 a사용
void sub1()
{
    a += 100;
    cout << "\n sub1의 a (file2.cpp) ==> " << a;
}
```

```
// file3.cpp
#include<iostream>
using namespace std;
static int b = 20; // file1에 선언된 변수 b와 별개로
                  // 사용하기 위해 static 추가
void sub2()
{
    b += 100;
    cout << "\n sub2의 b (file3.cpp) ==> " << b;
}
```

● 레지스터변수

- 변수 선언 시 자료형 앞에 register라는 예약어를 덧붙여 레지스터 변수로 선언
- 레지스터 변수는 자동변수와 모든 면에서 동일
- 자동변수가 스택에 기억공간 할당, 레지스터 변수는 CPU 내의 레지스터에 할당
→ 이유는 스택에 접근하는 것보다 빠르기 때문
- CPU 내의 레지스터는 제한 있음
- 기억 클래스 register를 붙이더라도 레지스터가 여분이 없으면 스택에 값 저장

레지스터 변수 사용

```
01 #include <iostream>
02 using namespace std;
03 long int power(register int x, register int n);
04 void main()
05 {
06     int a = 2, b = 5;
07     cout<<a<< " ^ " <<b<<" => "<<power(a, b)<<"\n";
08 }
09
10 long int power(register int x, register int n)
11 {
12     register int k;
13     long int p=1;
14     for(k =1; k<=n; k++)
15         p*=x;
16     return p;
17 }
```

