Storage Classes

- **Storage class specifiers : static, register, auto, extern**
  - Storage duration – how long an object exists in memory
  - Scope – where object can be referenced in program
  - Linkage – specifies the files in which an identifier is known
- **Automatic storage**
  - Object created and destroyed within its block
  - `auto`: default for local variables

```
auto double x, y;
```

  - `register`: tries to put variable into high-speed registers
    - Can only be used for automatic variables

Automatic Storage

- Object created and destroyed within its block
- `auto`: **default for local variables**

```
auto double x, y; //same as double x, y
```

- **Conserving memory**
  - because automatic variables exist only when they are needed.
  - They are created when the function in which they are defined is entered
  - and they are destroyed when the function is exited
- **Principle of least privilege**
  - Allowing access to data only when it is absolutely needed.
  - Why have variables stored in memory and accessible when in fact they are not needed?

Register Storage

- The storage-class specifier **register** can be placed before an automatic variable declaration
  - To suggest that the compiler maintain the variable in one of the computer's high-speed hardware registers.

```
register int counter;
```

  - If intensely used variables such as counters or totals can be maintained in hardware registers
- **Often, register declarations are unnecessary**
  - Today's optimizing compilers are capable of recognizing frequently used variables
  - Can decide to place them in registers without the need for a register declaration

Static storage Classes

- Variables exist for entire program execution
- Default value of zero
- `static`: local variables defined in functions.
  - ***Keep value after function ends***
  - Only known in their own function
- `extern`: default for ***global variables*** and functions
  - Known in any function

Tips for Storage Class

- **Defining a variable as global rather than local**
  - Allows unintended side effects to occur
  - When a function that does not need access to the variable accidentally or maliciously modifies it
- **In general, use of global variables** should be avoided : except in certain situations
- **Variables used only in a particular function**
  - Should be defined as local variables in that function
  - Rather than as external variables.

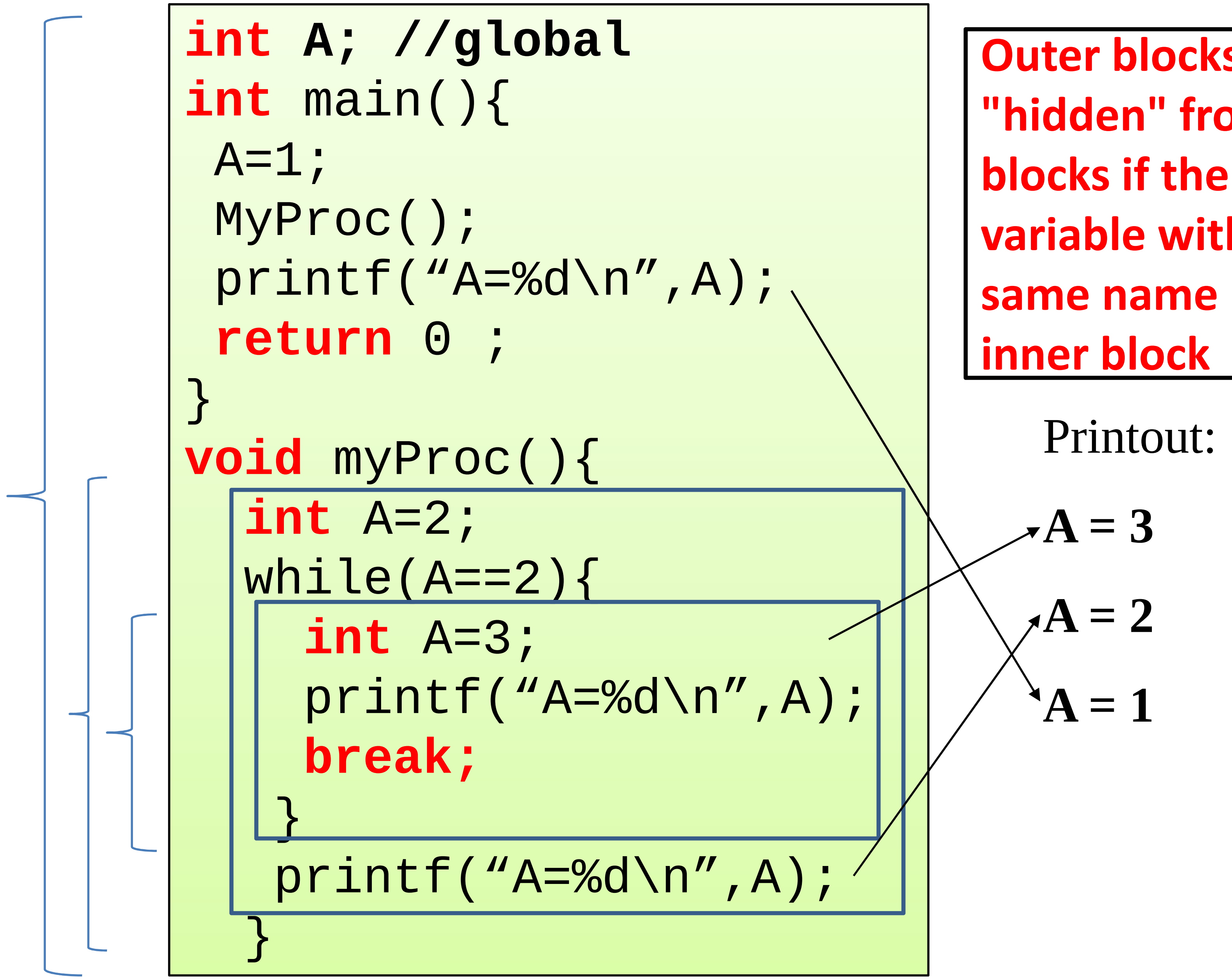
Scope Rules

- File scope
  - Identifier defined outside function, known in all functions
  - Used for *global variables, function definitions, function prototypes*
- Function scope
  - Can only be referenced inside a function body

Scope Rules

- Block scope
  - Identifier declared inside a block
    - Block scope begins at definition, ends at right brace
  - Used for *variables, function parameters (local variables of function*)
  - **Outer blocks "hidden" from inner blocks if there is a variable with the same name in the inner block**
- Function prototype scope
  - Used for identifiers in *parameter list*

Scope Rule Example



```
int A; //global
int main(){
 A=1;
 MyProc();
 printf("A=%d\n",A);
 return 0 ;
}
void myProc(){
  int A=2;
  while(A==2){
    int A=3;
    printf("A=%d\n",A);
    break;
   }
   printf("A=%d\n",A);
  }
```

Printout:

A = 3

A = 2

A = 1

Outer blocks "hidden" from inner blocks if there is a variable with the same name in the inner block

Scope and Life : Static Vs Global

```
int GA; //global
int main(){
 int i;
 GA=1;
 for(i=1;i<10;i++)
    MyProc();
 printf(“GA=%d”,GA);
 return 0 ;
}
void myProc(){
  static int SA=2;
  SA=SA+1;
}
```

Both SA and GA Variables exist for entire program execution

- SA initialized once
- SA can be accessible from myProc only
- But GA accessible from any part of Program

Scope Rule Example

```
int FunA(){return 4;}; //global
int main(){

 {
 int FunA(){return 3;};
 pintf(“FA=%d\n”,FunA());
 }


 pintf(“FA=%d\n”,FunA());
 return 0 ;
}
```

Outer blocks "hidden" from inner blocks if there is a variable with the same name in the inner block

Printout:

```
FA = 3

FA = 4
```

Compile using gcc

This code will not compile using c++/g++ compiler