The Three Attributes of an Identifier

Identifiers have three essential attributes:

- storage duration
- scope
- linkage

Storage Duration

storage duration

determines when memory is set aside for the variable and when that memory is released

automatic

allocated when the surrounding block is executed

deallocated when the block terminates

static

stays in the same storage location as long as the program is running

can retain its value indefinitely (until program terminates)

Automatic Storage Duration

automatic

allocated when the surrounding block is executed

deallocated when the block terminates

```
...
void Sort(int list[], int Sz) {

   int startIdx = 0;

   ...
}
```

declared inside a block

created (memory allocated) on each call

initialized on each call

deallocated (memory reclaimed) when call ends

Static Storage Duration

static

stays in the same storage location as long as the program is running

can retain its value indefinitely (until program terminates)

```
int numCallsToSort = 0;

...


void Sort(int list[], int Sz) {

   static int numSwaps = 0;

   ...
}
```

declared outside all blocks

initialized once, keeps its value until program ends

declared inside a block, with keyword `static`

initialized once, keeps its value from one call to the next

Scope

scope

(of an identifier) the range of program statements within which the identifier is recognized as a valid name

block scope

visible from its point of declaration to the end of the enclosing block

place declaration within a block

file scope

visible from its point of declaration to the end of the enclosing file

place declaration outside of all blocks (typically at beginning of file)

Block Scope

block scope

visible from its point of declaration to the end of the enclosing block

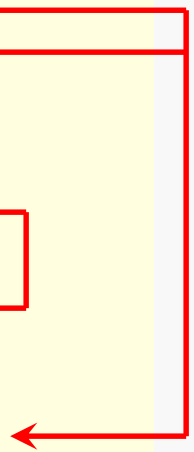
place declaration within a block

```
void Sort(int list[], int Sz) {

   static int numSwaps = 0;
   int startIdx = 0;

   for ( ... ) {
      int stopIdx = ...;

   }
   ...
   return;
}
```



declared inside a block

can only be referred to from declaration to end of block

File Scope

file scope

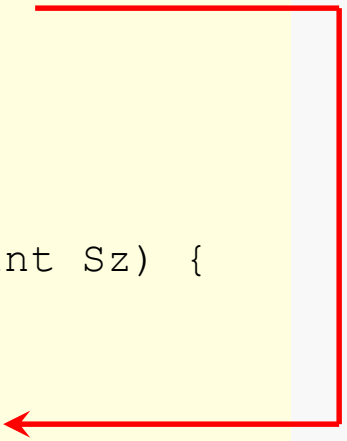
visible from its point of declaration to the end of the enclosing file

place declaration outside of all blocks (typically at beginning of file)

```
int numCallsToSort = 0;

...


void Sort(int list[], int Sz) {
...
}
...
```



declared outside all blocks

can be referred to from any function within the file

potentially dangerous

avoid unless necessary

pass parameters instead

Linkage

linkage

determines the extent to which the variable can be shared by different parts of the program

external linkage

may be shared by several (or all) files in the program

internal linkage

restricted to a single file, but shared by all functions within that file

no linkage

restricted to a single function

External Linkage

external linkage

may be shared by several (or all) files in the program

```
int numCallsToSort = 0;
...

void Sort(int list[], int Sz) {
...
}
```

declared outside all blocks

can be referred to from other files

potentially very dangerous

use only if necessary

Internal Linkage

internal linkage

restricted to a single file, but shared by all functions within that file

```
static int numCallsToSort = 0;
...

void Sort(int list[], int Sz) {
...
}
```

declared outside all blocks, using reserved word static

cannot be referred to from other files

potentially dangerous

use only if necessary

No Linkage

no linkage

restricted to a single function

```
...
void Sort(int list[], int Sz) {

   static int numSwaps = 0;
   int startIdx = 0;

   ...
}
```

declared inside a block

can only be referred to within the block where the declaration is placed

Determining the Attributes

The default storage duration, scope and linkage of a variable depend on the location of its declaration:

inside a block

automatic storage duration, block scope, no linkage

outside any block

static storage duration, file scope, external linkage

When the defaults are not satisfactory, see:

```
auto
static
extern
register
```

Example: Declarations/Definitions


```
// A.c
. . .

int externalA = 10;          // definition: external linkage,
                             //    static storage, file scope
static int staticA = 20;     // definition: internal linkage,
                             //    static storage, file scope
void A() {
   . . .

   int localA = 0;           // definition: no linkage,
   . . .
   static int localstaticA = 0; // definition: no linkage,
                                //    static storage, block scope
   . . .
}
```

```
// main.c
. . .

extern int externalA;   // declaration, not definition

int main() {
   . . .
}
```

Example: Full Source


```
// A.c
#include <stdio.h>

int externalA = 10;          // definition: external linkage,
                             //    static storage, file scope
static int staticA = 20;     // definition: internal linkage,
                             //    static storage, file scope
void A() {

   printf("   A():  externalA = %d\n", externalA);
   printf("   A():  staticA = %d\n", staticA);

   int localA = 0;           // definition: no linkage,
                             //    auto storage, block scope
   ++localA;
   printf("   A():  localA = %d\n", localA);

   static int localstaticA = 0; // definition: no linkage,
                                //    static storage, block scope
   ++localstaticA;
   printf("   A():  localstaticA = %d\n", localstaticA);
}
```

Example: Full Source

```
// main.c
#include <stdio.h>
#include "A.h"

extern int externalA;   // declaration, not definition

int main() {

   printf("main():  making first call to A():\n");
   A();
   printf("main():  resetting externA:\n");
   externalA = 5;
   printf("main():  making second call to A():\n");
   A();
   printf("main():  making third call to A():\n");
   A();

   return 0;
}
```

```
// A.h
#ifndef A_H
#define A_H

void A();

#endif
```

Example: First Call


```
printf("main():  making first call to A():\n");
A();
. . .
```

```
main():  making first call to A():
   A():  externalA = 10
   A():  staticA = 20
   A():  localA = 1
   A():  localstaticA = 1
```

Example: Access and Second Call


```
. . .
printf("main():  resetting externA:\n");
externalA = 5;
printf("main():  making second call to A():\n");
A();
. . .
```

```
main():  making second call to A():
   A():  externalA = 5
   A():  staticA = 20
   A():  localA = 1
   A():  localstaticA = 2
```

Example: Third Call

```
. . .
printf("main():  making third call to A():\n");
A();
```

```
main():  making third call to A():
   A():  externalA = 5
   A():  staticA = 20
   A():  localA = 1
   A():  localstaticA = 3
```