Static and Global Variables

Prof. Jyotiprakash Mishra
mail@jyotiprakash.org

Global Variables

Program 1:

```c
#include <stdio.h>
int counter = 0;
void increment() {
  counter++;
}
void display() {
  printf("Counter: %d\n", counter);
}
int main() {
  display();
  increment();
  increment();
  display();
  counter = 100;
  display();
  return 0;
}
```

Output:

```
Counter: 0
Counter: 2
Counter: 100
```

Note:

Global variable accessible from all functions in the file. Initialized once at program start. Persists for entire program lifetime.

Static Local Variables

Program 2:

```c
#include <stdio.h>
void count_calls() {
  static int calls = 0;
  calls++;
  printf("Function called %d times\n",
    calls);
}
int main() {
  count_calls();
  count_calls();
  count_calls();
  count_calls();
  return 0;
}
```

Output:

```
Function called 1 times
Function called 2 times
Function called 3 times
Function called 4 times
```

Note:

Static local variable retains value between function calls. Initialized only once. Has local scope but lifetime of entire program.

Static vs Auto Variables

Program 3:

```c
#include <stdio.h>
void test_auto() {
  int x = 0;
  x++;
  printf("Auto: %d\n", x);
}
void test_static() {
  static int y = 0;
  y++;
  printf("Static: %d\n", y);
}
int main() {
  test_auto();
  test_auto();
  test_static();
  test_static();
  return 0;
}
```

Output:

```
Auto: 1
Auto: 1
Static: 1
Static: 2
```

Note:

Auto (default) variables reset each call. Static variables persist. Auto on stack, static in data segment. Different lifetimes.

Static Global Variables

file1.c:

```c
#include <stdio.h>
static int secret = 42;
void show_secret() {
  printf("Secret: %d\n", secret);
}
int main() {
  show_secret();
  return 0;
}
```

file2.c:

```c
#include <stdio.h>
extern int secret;
void access_secret() {
  printf("Secret: %d\n", secret);
}
```

Output:

```
Secret: 42
```

Compile:

```
gcc file1.c file2.c
Error: undefined reference to secret
```

Note:

```
static global variable has file
scope only. Cannot be accessed from
other files even with extern.
Internal linkage.
```

Extern Keyword Basics

global.c:

```c
int shared = 100;
```

main.c:

```c
#include <stdio.h>
extern int shared;
int main() {
  printf("Shared: %d\n", shared);
  shared = 200;
  printf("Shared: %d\n", shared);
  return 0;
}
```

Output:

```
Shared: 100
Shared: 200
```

Compile:

```
gcc global.c main.c -o prog
./prog
```

Note:

```
extern declares variable defined
elsewhere. One definition in
global.c, declaration in main.c.
External linkage.
```

Multiple Extern Declarations

data.c:

```c
int count = 0;
float value = 3.14;
```

func1.c:

```c
extern int count;
void increment() {
  count++;
}
```

func2.c:

```c
#include <stdio.h>
extern int count;
extern float value;
void display() {
  printf("Count: %d, Value: %.2f\n",
    count, value);
}
```

main.c:

```c
extern void increment();
extern void display();
int main() {
  display();
  increment();
  increment();
  display();
  return 0;
}
```

Output:

```
Count: 0, Value: 3.14
Count: 2, Value: 3.14
```

Note:

Multiple files can extern same variable. One definition in data.c, declarations in func1.c and func2.c. All share same memory location.

Extern in Header Files

config.h:

```
#ifndef CONFIG_H
#define CONFIG_H
extern int max_users;
extern char app_name[];
#endif
```

config.c:

```
int max_users = 100;
char app_name[] = "MyApp";
```

main.c:

```
#include <stdio.h>
#include "config.h"
int main() {
  printf("App: %s\n", app_name);
  printf("Max users: %d\n", max_users);
  max_users = 200;
  printf("Max users: %d\n", max_users);
  return 0;
}
```

Output:

```
App: MyApp
Max users: 100
Max users: 200
```

Note:

```
Common pattern: extern declarations
in header, definitions in source.
Header can be included in multiple
files safely.
```

Static Function Scope

helper.c:

```c
#include <stdio.h>
static void private_func() {
  printf("Private function\n");
}
void public_func() {
  printf("Public function\n");
  private_func();
}
```

main.c:

```c
extern void public_func();
extern void private_func();
int main() {
  public_func();
  return 0;
}
```

Output:

```
Public function
Private function
```

Compile:

```
gcc helper.c main.c
Error: undefined reference to
private_func
```

Note:

```
static function has file scope.
Cannot be called from other files.
Encapsulation in C. Remove extern
call to compile successfully.
```

Initialization of Static Variables

Program 9:

```c
#include <stdio.h>
int global_init = 10;
static int static_global = 20;
void test() {
  static int static_local = 30;
  int auto_var;
  printf("Global: %d\n", global_init);
  printf("Static global: %d\n",
    static_global);
  printf("Static local: %d\n",
    static_local);
  printf("Auto (garbage): %d\n",
    auto_var);
  static_local++;
}
int main() {
  test();
  test();
  return 0;
}
```

Output:

```
Global: 10
Static global: 20
Static local: 30
Auto (garbage): -1234567
Global: 10
Static global: 20
Static local: 31
Auto (garbage): 98765
```

Note:

```
Static variables initialized to 0
if not explicitly initialized.
Auto variables have garbage values.
Static init happens once.
```

Global Variable Shadowing

Program 10:

```c
#include <stdio.h>
int x = 100;
void test() {
  int x = 200;
  printf("Local x: %d\n", x);
  {
    int x = 300;
    printf("Block x: %d\n", x);
  }
  printf("Local x: %d\n", x);
}
int main() {
  printf("Global x: %d\n", x);
  test();
  printf("Global x: %d\n", x);
  return 0;
}
```

Output:

```
Global x: 100
Local x: 200
Block x: 300
Local x: 200
Global x: 100
```

Note:

```
Local variables shadow global ones.
Inner scope hides outer scope.
Global x unchanged. Each scope has
its own x variable.
```

Storage Class Specifiers

Program 11:

```c
#include <stdio.h>
int global;
static int static_global;
extern int external;
void test() {
  auto int local = 10;
  static int static_local = 20;
  register int fast = 30;
  printf("Auto: %d\n", local);
  printf("Static: %d\n", static_local);
  printf("Register: %d\n", fast);
}
int main() {
  test();
  return 0;
}
```

Output:

```
Auto: 10
Static: 20
Register: 30
```

Note:

```
Storage classes: auto (default),
static, extern, register.
auto: automatic storage.
static: persistent storage.
register: hint for CPU register.
extern: external linkage.
```

Extern with Functions

math_ops.c:

```c
int add(int a, int b) {
  return a + b;
}
int multiply(int a, int b) {
  return a * b;
}
```

main.c:

```c
#include <stdio.h>
extern int add(int, int);
extern int multiply(int, int);
int main() {
  printf("Add: %d\n", add(5, 3));
  printf("Multiply: %d\n",
    multiply(5, 3));
  return 0;
}
```

Output:

```
Add: 8
Multiply: 15
```

Note:

Functions have external linkage by default. extern keyword optional for functions. Usually put in header files.

Const Global Variables

constants.c:

```c
const int MAX = 100;
const float PI = 3.14159;
```

main.c:

```c
#include <stdio.h>
extern const int MAX;
extern const float PI;
int main() {
  printf("MAX: %d\n", MAX);
  printf("PI: %.5f\n", PI);
  return 0;
}
```

Output:

```
MAX: 100
PI: 3.14159
```

Note:

```
const variables can be shared via
extern. Declared const in both
definition and declaration.
Read-only across files.
```

Static Counter Pattern

Program 14:

```c
#include <stdio.h>
int* get_counter() {
  static int counter = 0;
  counter++;
  return &counter;
}
int main() {
  int *p1 = get_counter();
  int *p2 = get_counter();
  int *p3 = get_counter();
  printf("Counter: %d\n", *p1);
  printf("Counter: %d\n", *p2);
  printf("Counter: %d\n", *p3);
  printf("Same address: %d\n",
    p1 == p2 && p2 == p3);
  return 0;
}
```

Output:

```
Counter: 3
Counter: 3
Counter: 3
Same address: 1
```

Note:

Safe to return pointer to static variable (unlike auto variables). Static persists after function returns. All pointers reference same static variable.

Global Array Sharing

data.c:

```c
int numbers[5] = {1, 2, 3, 4, 5};
```

utils.c:

```c
#include <stdio.h>
extern int numbers[];
void print_array() {
  int i;
  for (i = 0; i < 5; i++) {
    printf("%d ", numbers[i]);
  }
  printf("\n");
}
```

main.c:

```c
extern void print_array();
extern int numbers[];
int main() {
  print_array();
  numbers[0] = 100;
  print_array();
  return 0;
}
```

Output:

```
1 2 3 4 5
100 2 3 4 5
```

Note:

```
Arrays can be shared via extern.
Size not required in extern
declaration. All files access
same array in memory.
```

Static Initialization Order

Program 16:

```c
#include <stdio.h>
int a = 10;
int b = a + 5;
static int c = 20;
static int d = c + 10;
int main() {
  printf("a: %d\n", a);
  printf("b: %d\n", b);
  printf("c: %d\n", c);
  printf("d: %d\n", d);
  return 0;
}
```

Output:

```
Compilation error: initializer
element is not constant
```

Note:

```
Global/static variables must be
initialized with constant
expressions. Cannot use variables
in initialization. Must be known
at compile time.
```

Static String Literal

Program 17:

```c
#include <stdio.h>
const char* get_message() {
  static const char* msg = "Hello";
  return msg;
}
char* get_local() {
  char* local = "World";
  return local;
}
int main() {
  printf("%s\n", get_message());
  printf("%s\n", get_local());
  const char* p1 = get_message();
  const char* p2 = get_message();
  printf("Same: %d\n", p1 == p2);
  return 0;
}
```

Output:

```
Hello
World
Same: 1
```

Note:

```
String literals have static storage.
Both static and auto pointers to
string literals are safe. Static
ensures same pointer returned.
```

Multiple Definition Error

file1.c:

```c
int shared = 100;
```

file2.c:

```c
int shared = 200;
```

main.c:

```c
#include <stdio.h>
extern int shared;
int main() {
  printf("Shared: %d\n", shared);
  return 0;
}
```

Compile:

```
gcc file1.c file2.c main.c
Error: multiple definition of shared
```

Fix:

```
Define in one file only.
Declare extern in others.
OR use static in one to make it
file-local.
```

Note:

```
Only one definition allowed across
all files. Multiple extern
declarations OK.
```

Tentative Definitions

file1.c:

```c
int x;
```

file2.c:

```c
int x;
```

main.c:

```c
#include <stdio.h>
extern int x;
int main() {
  x = 42;
  printf("x: %d\n", x);
  return 0;
}
```

Output:

```
x: 42
```

Note:

```
Declaration without initialization
is tentative definition. Allowed in
multiple files. Linker merges them.
Better to use extern explicitly.
```

Header Guard with Externs

globals.h:

```
#ifndef GLOBALS_H
#define GLOBALS_H
extern int app_version;
extern char app_name[];
extern void init_app();
#endif
```

globals.c:

```
#include <stdio.h>
int app_version = 1;
char app_name[] = "MyApp";
void init_app() {
  printf("Initializing %s v%d\n",
    app_name, app_version);
}
```

main.c:

```
#include "globals.h"
int main() {
  init_app();
  return 0;
}
```

Output:

```
Initializing MyApp v1
```

Note:

Standard pattern: extern in header, definitions in source. Header can be included multiple times safely. Include guards prevent redefinition.