CSC 4630

Supplementary Material on C

Storage Classes
Storage Classes Overview

- Alternatives for storing data

- Storage Classes:
  - Automatic
  - Register
  - External
  - Static

- Scope: Statements which can reference a variable

- Life: Length of time the contents are valid
Storage Classes

A **declaration** of a variable is an announcement of its type. It is analogous to a function **prototype**. A variable may be declared more than once in a program package.

A **definition** of a variable is the allocation of storage for it. This also serves as a declaration. It is analogous to a function definition. A variable can't be defined more than once in a program.
AUTOMATIC IDENTIFIERS — SCOPE

SCOPE:

THE SCOPE OF AN IDENTIFIER IS THE PART OF A PROGRAM OVER WHICH THE IDENTIFIER IS DEFINED

FOR AUTOMATIC IDENTIFIERS, THIS IS THE COMPOUND STATEMENT OR BLOCK IN WHICH IT WAS DECLARED
Automatic Storage Class

• Scope: Local, block
• Life: Created when block entered
  Becomes non-existent when block exited
• Inner declarations supersede outer ones

```c
int main(void)
{
    void sub(int);
    int x; /* Known only in main() */
    x = getchar();
    if (x == 'c') {
        float x;
        /* Known in this block */
        x = .75 /* refers to x in line 08 */
        ...
    } else ...
    sub(100); /* return */
}

void sub(int x) /* This x known only in sub() */
{
    printf("%d\n", x*x*x); /* return */
}
```
Automatic Storage Class - better style

- **Scope:** Local, block
- **Life:** Created when block entered
  Becomes non-existent when block exited
- **Inner declarations** supersede outer ones

```c
01  int main(void)
02  {
03      void sub(int);
04      int x;           /* Known only in main() */
05
06      x = getchar();
07      if (x == 'c') {
08          float y;       /* Known in this block */
09          y = .75;
10          ...
11          ...
12      }
13      sub(100);       /* return */
14  }
15
16  void sub(int z)     /* This z known only in sub() */
17  {
18      printf("%d\n", z*z*z);
19  }
```
Registered Storage Class

- Speed and efficiency
- Machine registers used instead of stack
- Function parameters and automatics
- Scope and life: same as automatics
- Number available is system-dependent
- Data types: usually char, int, pointer

```
int f(register int count, num)
{
    register int i;
    ...
}
```
External Storage Class

- Global
- Variables defined outside a function, initialized to 0
- Scope: From definition to end of file declaration
- Life: Life of program
- Advantage: Data sharing

```c
01   int status;
02   char sysname[20];
03   void fillbuf(void), emptybuf(void);
04
05   int main(void)
06     {
07       fillbuf();
08       ...
21       emptybuf();
22       ...
25     }
26
27   char buf[1024];
28
29   void fillbuf(void)
30     {
31       ...
38     }
39
40   void emptybuf(void)
41     {
42       ...
49     }
```
Multiple Source Files

- C programs often kept in several files
- Keyword `extern` used to access external variables defined in another file

```
one.c
two.c

int status;
char sysname[20];
void fillbuf( void ), emptybuf( void );

int main( void )
{
    fillbuf( );
    ... if ( expression )
        emptybuf( );
    ...
}

extern int status;
extern char sysname[];
char buf[1024];

void fillbuf( void )
{
    ...
}

void emptybuf( void )
{
    ...
}

$ gcc -o wholeprogram one.c two.c
```
STATIC IDENTIFIERS — SCOPE

INTERNAL:

"LOCAL" TO FUNCTION

EXTERNAL:

SOURCE FILE IN WHICH DEFINED AND NO OTHER FILE
STATIC IDENTIFIERS — LIFE

INTERNAL:

LIFE OF PROGRAM

EXTERNAL:

LIFE OF PROGRAM
Static Storage Class

- **Scope:**
  - **External:** Declaration to end-of-file
    Not available to other files
  - **Internal:** Function, block

- **Life:** Life of program

- **Value:** Defaults to 0

- **Advantages:** Permanence and privacy

```c
static int count;

void g(int amnt) {
    static int total = 1024;
    int x = 0;
    total -= amnt;
}

/* A static function is private */
/* to its source file */
static errmsg(void) {
    ...
}
```
ORGANIZATION OF PROGRAM FILES

projX.h

#define TABLESIZE 1000
#define SYSNAMELEN 20
#define MASK 010
#define MAX(A,B) (a>b?a:b)
extern int status;
extern char sysname[SYSNAMELEN];
extern double table[];
extern void calc_err();
extern double std_dev();
typedef char byte;

main.c

#include "projX.h"
main()

bufctl.c

#include <stdio.h>
#include "projX.h"
static byte buf[1024];
fillbuf()
emptybuf()

defs.c

#include "projX.h"
int status;
char sysname[SYSNAMELEN];
double table[TABLESIZE];

calc.c

double std_dev()

void calc_err()

(prototyped not ANSI)
calc.h
#define NUMBER '0'
void push(double);
double pop(void);
int gettop(char []);
int getch(void);
void ungetch(int);

main.c:
#include <stdio.h>
#include <stdlib.h>
#include "calc.h"
#define MAXOP 100
main() {
    ...
}

getop.c:
#include <stdio.h>
#include <ctype.h>
#include "calc.h"
getop() {
    ...
}

getch.c:
#include <stdio.h>
#define BUFSIZE 100
char buf[BUFSIZE];
int bufp = 0;
int getch(void) {
    ...
}
void ungetch(int) {
    ...
}

stack.c:
#include <stdio.h>
#include "calc.h"
#define MAXVAL 100
int sp = 0;
double val[MAXVAL];
void push(double) {
    ...
}
double pop(void) {
    ...
}
Where Variables are Stored

(Hypothetical System)

Instructions

Data Area
- externals and statics
- string constants

Stack
- automatic variables
- saved registers
- function arguments
- automatic variables
- saved registers
- main()’s arguments
- system-dependent info

- External and Static Variables
  Life of program
  Values default to 0
  Initialized before execution

- Automatic Variables
  Temporary, block life
  Default value undefined
IDENTIFIER INITIALIZATION

AUTOMATIC IDENTIFIERS:

WITHIN DECLARATION

WITHIN ASSIGNMENT STATEMENT

CANNOT!  ARRAYS MAY NOT BE INITIALIZED IN SINGLE STATEMENT

NO DEFAULT VALUES
Variable Initialization

- **Automatic Variables**
  Default value undefined
  Can initialize arrays, structures in declaration

- **External and Static Variables**
  Default value is 0
  Can initialize arrays, structures in declaration

```c
1 char alpha [10];
2 char beta [10] = {'a', 'b', 'c'};
3 char gamma [] = "This is gamma";
4 char delta [5][10] = {
    "line 1",
    "line 2",
    "line 3"};

7

8 int num1 [10];
9 int num2 [10] = {2, 4, 6, 8, 10};
10 int num3 [5][10] = {
    {0, 1, 2, 3, 4},
    {2, 4, 6, 8, 10},
    {3, 6, 8, 10, 12}};

13
14 main()
15 {
16     static char local [] = "local string";
17
18 }
```
## Storage Class Summary

<table>
<thead>
<tr>
<th>Class</th>
<th>Scope</th>
<th>Life</th>
<th>Storage</th>
<th>Initialize Arrays, Structs</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>automatic</td>
<td>block</td>
<td>block active</td>
<td>stack</td>
<td><strong>no</strong></td>
<td>undefined</td>
</tr>
<tr>
<td>register</td>
<td>block</td>
<td>block active</td>
<td>machine register</td>
<td><strong>yes</strong></td>
<td>undefined</td>
</tr>
<tr>
<td>external</td>
<td>declaration to end-of-file</td>
<td>permanent</td>
<td>data area</td>
<td>yes</td>
<td>0</td>
</tr>
<tr>
<td>static external</td>
<td>declaration to end-of-file</td>
<td>permanent</td>
<td>data area</td>
<td>yes</td>
<td>0</td>
</tr>
<tr>
<td>static internal</td>
<td>block</td>
<td>permanent</td>
<td>data area</td>
<td>yes</td>
<td>0</td>
</tr>
</tbody>
</table>

1. Speed advantage
2. For function arguments, value passed
3. Can be accessed from other files
4. Can not be accessed from other files
ANSI:

Could be done
in automatic too

IDENTIFIER INITIALIZATION — EXAMPLE

1: char alpha [10];
2: char beta [10] = { 'a', 'b', 'c' };  \text{ (separator) }
3: char gamma [] = "This is gamma";
4: char delta [5][10] = {
5:     "line 1",
6:     "line 2",
7:     "line 3"};
8: int num1 [10];
9: int num2 [10] = {2, 4, 6, 8, 10};
10: int num3 [5][10] = {
11:     {0, 1, 2, 3, 4},
12:     {2, 4, 6, 8, 10},
13:     {3, 6, 8, 10, 12}};
14: 
15: main()
16: {
17:     .
18:     .
19:     .
20:     static char local [] = "local string";
21:     .
22:     .
23: }