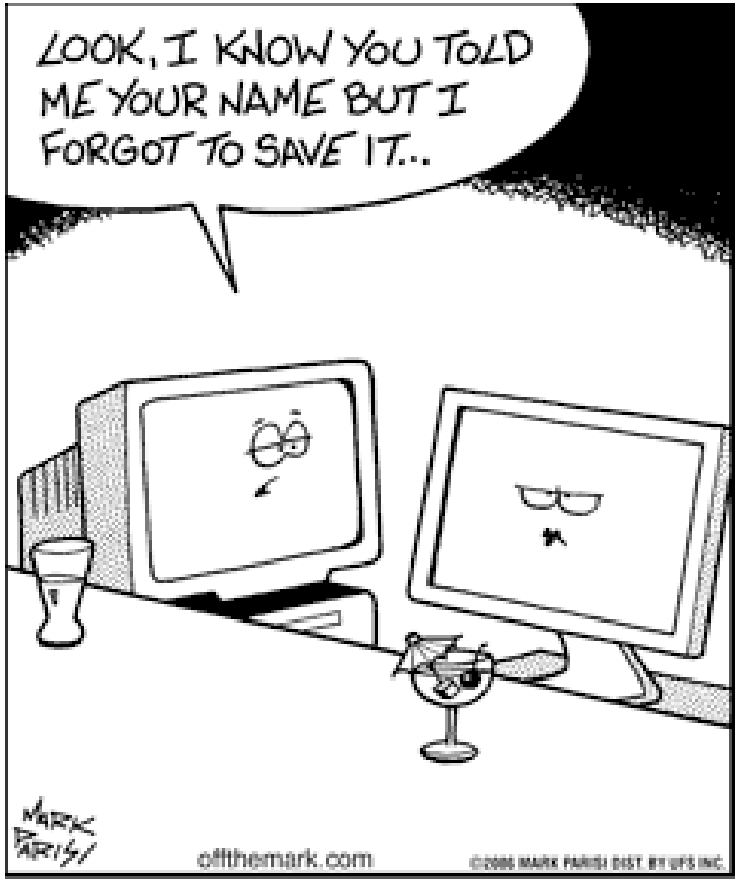
CSE101-lec 14

Storage Classes and Scope Rules


LOVELY PROFESSIONAL UNIVERSITY
PUNJAB (INDIA)
L
P
U

Outline

- Storage Classes
  - auto
  - static
  - extern
  - register
- Scope Rules



Storage Classes

- Storage class specifies
  - Storage duration – how long the variable retains a particular value
  - Scope – or visibility of the variable i.e the portion of the program within which the variables are recognized.

Storage Classes: Auto

- Automatic storage
  - `auto int x, y;`
  - It is the default storage class
- Storage – **Memory.**
- Default initial value – An unpredictable value, which is often called a **garbage value.**
- Scope – **Local to the block** in which the variable is defined.
- Life – Till the control remains within the block in which the variable is defined.

Storage Classes: Register

- `register`: tries to put variable into high-speed registers.
  - `register int counter = 1;`

- Storage - **CPU registers**.
- Default initial value - **Garbage value.**
- Scope - **Local** to the block in which the variable is defined.
- Life - Till the control remains within the block in which the variable is defined.

Storage Classes: Static

- Static storage
  - Storage – **Memory.**
  - Default initial value – **Zero.**
  - Scope – **Local** to the block in which the variable is defined.
  - Life – variable will retain throughout the program

Storage Classes: extern

Default for global variables and functions

- Known in any function

- Storage – **Memory**.
- Default initial value – **Zero**.
- Scope – **Global**.
- Life – As long as the program's execution doesn't come to an end.

Scope Rules

- The *scope* of a variable is the portion of a program where the variable has meaning (where it exists).
- A global variable has global (unlimited) scope.
- A local variable's scope is restricted to the function that declares the variable.
- A block variable's scope is restricted to the block in which the variable is declared.

Local variables

- Parameters and variables declared inside the definition of a function are *local*.
- They only exist inside the function body.
- Once the function returns, the variables no longer exist!
  - That's fine! We don't need them anymore!

Block Variables

- You can also declare variables that exist only within the *body* of a compound statement *(a block)*:

```
{
int f;
    …
    …
}
```

Global variables

- You can declare variables outside of any function definition – these variables are *global variables*.
- Any function can access/change global variables.

Global Scope:
Cruising speed
Cruising Height

Local Scope:
Distance to destination

Block Scope:
Landing Gear state
Rudder tilt

Example code

```c
#include <stdio.h>

void useLocal( void );       /* function prototype */
void useStaticLocal( void ); /* function prototype */
void useGlobal( void );      /* function prototype */

int x = 1; /* global variable */

/* function main begins program execution */
int main()
{
   int x = 5; /* local variable to main */

   printf("local x in outer scope of main is %d\n", x );

   { /* start new scope */
      int x = 7; /* local variable to new scope */

      printf( "local x in inner scope of main is %d\n", x );
   } /* end new scope */

   printf( "local x in outer scope of main is %d\n", x );
```

This program demonstrates each storage class and scopes of variables

Example code

```
   useLocal();       /* useLocal has automatic local x */
   useStaticLocal(); /* useStaticLocal has static local x */
   useGlobal();      /* useGlobal uses global x */

   useLocal();       /* useLocal reinitializes automatic local x */
   useStaticLocal(); /* static local x retains its prior value */
   useGlobal();      /* global x also retains its value */

   printf( "local x in main is %d\n", x );

   return 0; /* indicates successful termination */

} /* end main */

/* useLocal reinitializes local variable x during each call */
void useLocal( void )
{
   int x = 25;  /* initialized each time useLocal is called */

   printf( "\nlocal x in a is %d after entering a\n", x );
   x++;
   printf( "local x in a is %d before exiting a\n", x );
} /* end function useLocal */
```

This program demon-strates each storage class and scopes of variables

Example code

```
/* useStaticLocal initializes static local variable x only the first time
 the function is called; value of x is saved between calls to this
    function */
void useStaticLocal( void )
{
   /* initialized only first time useStaticLocal is called */
   static int x = 50;

   printf( "\nlocal static x is %d on entering b\n", x );
   x++;
   printf( "local static x is %d on exiting b\n", x );
} /* end function useStaticLocal */

/* function useGlobal modifies global variable x during each call */
void useGlobal( void )
{
   printf( "\nglobal x is %d on entering c\n", x );
   x *= 10;
   printf( "global x is %d on exiting c\n", x );
}
```

This program demon-strates each storage class and scopes of variables

output

```
local x in outer scope of main is 5
local x in inner scope of main is 7
local x in outer scope of main is 5

local x in a is 25 after entering a
local x in a is 26 before exiting a

local static x is 50 on entering b
local static x is 51 on exiting b

global x is 1 on entering c
global x is 10 on exiting c

local x in a is 25 after entering a
local x in a is 26 before exiting a

local static x is 51 on entering b
local static x is 52 on exiting b

global x is 10 on entering c
global x is 100 on exiting c
local x in main is 5
```

```c
int main()
{
   int x = 5; /* local variable to main */

   printf("local x in outer scope of main is %d\n", x );

   { /* start new scope */
      int x = 7; /* local variable to new scope */

      printf( "local x in inner scope of main is %d\n", x );
   } /* end new scope */

   printf( "local x in outer scope of main is %d\n", x );
```

output

```
local x in outer scope of main is 5
local x in inner scope of main is 7
local x in outer scope of main is 5

local x in a is 25 after entering a
local x in a is 26 before exiting a

local static x is 50 on entering b
local static x is 51 on exiting b

global x is 1 on entering c
global x is 10 on exiting c

local x in a is 25 after entering a
local x in a is 26 before exiting a

local static x is 51 on entering b
local static x is 52 on exiting b

global x is 10 on entering c
global x is 100 on exiting c
local x in main is 5
```

```
void useLocal( void )
{
   int x = 25;  /* initialized each time useLocal is called
   printf( "\nlocal x in a is %d after entering a\n", x );
   x++;
   printf( "local x in a is %d before exiting a\n", x );
} /* end function useLocal */
```

output

```
local x in outer scope of main is 5
local x in inner scope of main is 7
local x in outer scope of main is 5

local x in a is 25 after entering a
local x in a is 26 before exiting a

local static x is 50 on entering b
local static x is 51 on exiting b

global x is 1 on entering c
global x is 10 on exiting c

local x in a is 25 after entering a
local x in a is 26 before exiting a

local static x is 51 on entering b
local static x is 52 on exiting b

global x is 10 on entering c
global x is 100 on exiting c
local x in main is 5
```

```
void useStaticLocal( void )
{
   /* initialized only first time useStaticLocal is call
   static int x = 50;


   printf( "\nlocal static x is %d on entering b\n", x )
   x++;
   printf( "local static x is %d on exiting b\n", x );
} /* end function useStaticLocal */
```

output

```
local x in outer scope of main is 5
local x in inner scope of main is 7
local x in outer scope of main is 5

local x in a is 25 after entering a
local x in a is 26 before exiting a

local static x is 50 on entering b
local static x is 51 on exiting b

global x is 1 on entering c
global x is 10 on exiting c

local x in a is 25 after entering a
local x in a is 26 before exiting a

local static x is 51 on entering b
local static x is 52 on exiting b

global x is 10 on entering c
global x is 100 on exiting c
local x in main is 5
```

```
void useGlobal( void )
{
   printf( "\nglobal x is %d on entering c\n", x );
   x *= 10;
   printf( "global x is %d on exiting c\n", x );
} /* end function useGlobal */
```

output

```
local x in outer scope of main is 5
local x in inner scope of main is 7
local x in outer scope of main is 5

local x in a is 25 after entering a
local x in a is 26 before exiting a

local static x is 50 on entering b
local static x is 51 on exiting b

global x is 1 on entering c
global x is 10 on exiting c

local x in a is 25 after entering a
local x in a is 26 before exiting a

local static x is 51 on entering b
local static x is 52 on exiting b

global x is 10 on entering c
global x is 100 on exiting c
local x in main is 5
```

```
void useLocal( void )
{
   int x = 25;  /* initialized each time useLocal is called
   printf( "\nlocal x in a is %d after entering a\n", x );
   x++;
   printf( "local x in a is %d before exiting a\n", x );
} /* end function useLocal */
```

output

```
local x in outer scope of main is 5
local x in inner scope of main is 7
local x in outer scope of main is 5

local x in a is 25 after entering a
local x in a is 26 before exiting a

local static x is 50 on entering b
local static x is 51 on exiting b

global x is 1 on entering c
global x is 10 on exiting c

local x in a is 25 after entering a
local x in a is 26 before exiting a

local static x is 51 on entering b
local static x is 52 on exiting b

global x is 10 on entering c
global x is 100 on exiting c
local x in main is 5
```

```
void useStaticLocal( void )
{
   /* initialized only first time useStaticLocal is call
   static int x = 50;


   printf( "\nlocal static x is %d on entering b\n", x )
   x++;
   printf( "local static x is %d on exiting b\n", x );
} /* end function useStaticLocal */
```

output

```
local x in outer scope of main is 5
local x in inner scope of main is 7
local x in outer scope of main is 5

local x in a is 25 after entering a
local x in a is 26 before exiting a

local static x is 50 on entering b
local static x is 51 on exiting b

global x is 1 on entering c
global x is 10 on exiting c

local x in a is 25 after entering a
local x in a is 26 before exiting a

local static x is 51 on entering b
local static x is 52 on exiting b

global x is 10 on entering c
global x is 100 on exiting c
local x in main is 5
```

```
void useGlobal( void )
{
   printf( "\nglobal x is %d on entering c\n", x );
   x *= 10;
   printf( "global x is %d on exiting c\n", x );
} /* end function useGlobal */
```

output

```
local x in outer scope of main is 5
local x in inner scope of main is 7
local x in outer scope of main is 5

local x in a is 25 after entering a
local x in a is 26 before exiting a

local static x is 50 on entering b
local static x is 51 on exiting b

global x is 1 on entering c
global x is 10 on exiting c

local x in a is 25 after entering a
local x in a is 26 before exiting a

local static x is 51 on entering b
local static x is 52 on exiting b

global x is 10 on entering c
global x is 100 on exiting c
local x in main is 5
```

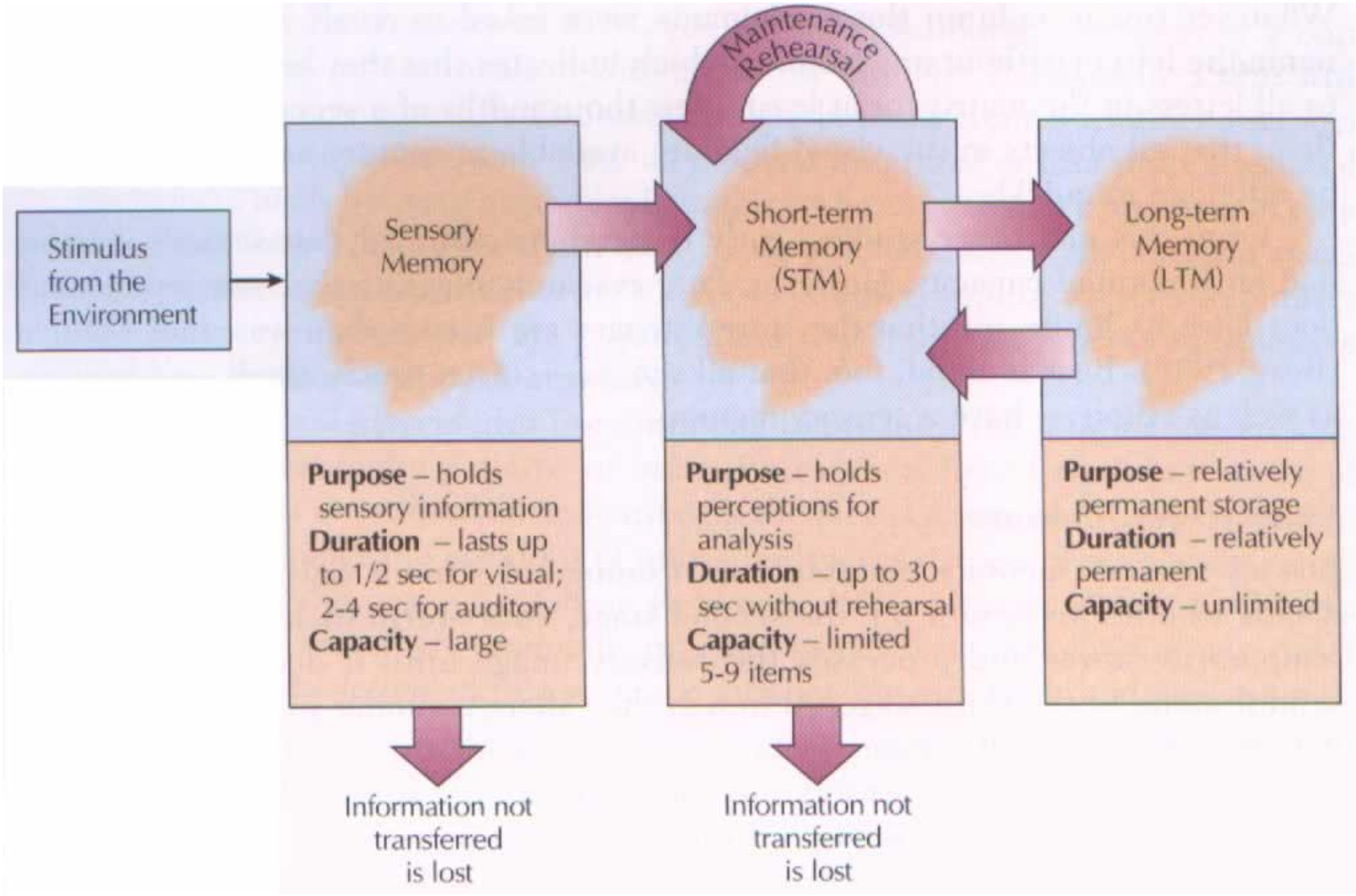
```
useLocal();       /* useLocal has automatic local x */
useStaticLocal(); /* useStaticLocal has static local x */
useGlobal();      /* useGlobal uses global x */
useLocal();       /* useLocal reinitializes automatic loca
useStaticLocal(); /* static local x retains its prior valu
useGlobal();      /* global x also retains its value */

printf( "local x in main is %d\n", x );

return 0; /* indicates successful termination */

/* end main */
```

Example


Maintenance Rehearsal
Stimulus from the Environment
Sensory Memory
Short-term Memory (STM)
Long-term Memory (LTM)
Purpose – holds sensory information
Duration – lasts up to 1/2 sec for visual; 2-4 sec for auditory
Capacity – large
Purpose – holds perceptions for analysis
Duration – up to 30 sec without rehearsal
Capacity – limited 5-9 items
Purpose – relatively permanent storage
Duration – relatively permanent
Capacity – unlimited
Information not transferred is lost
Information not transferred is lost