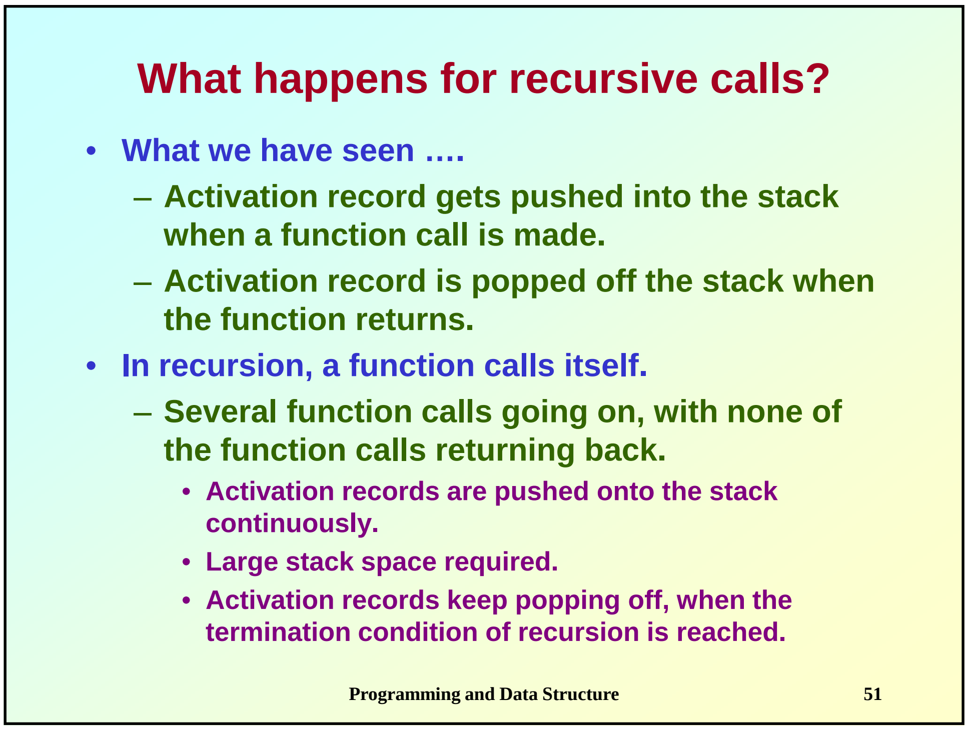
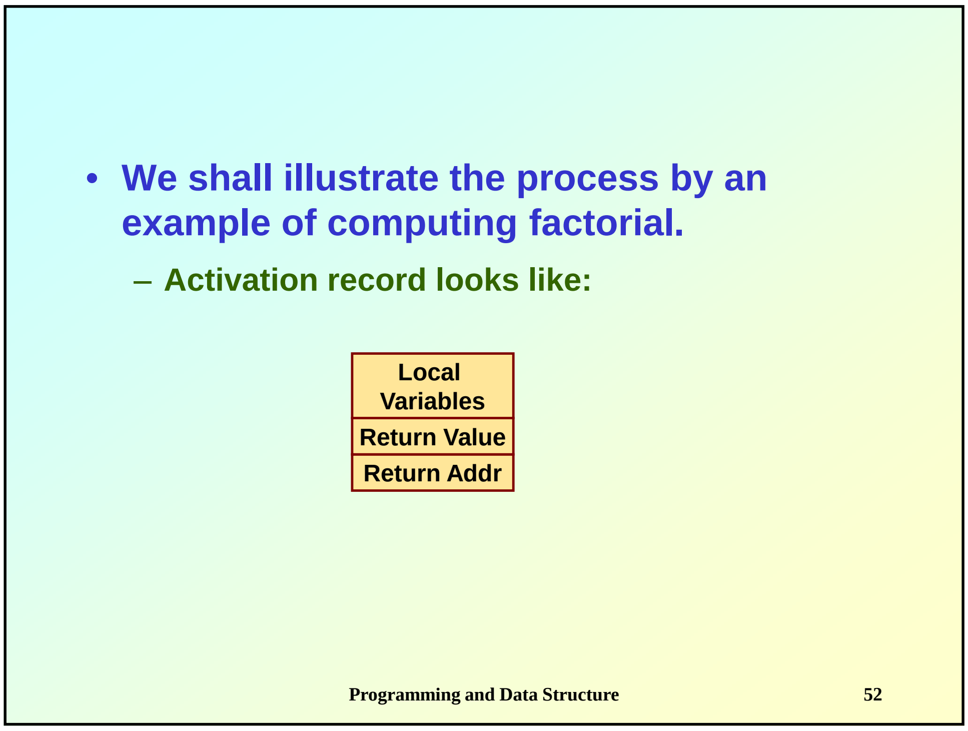


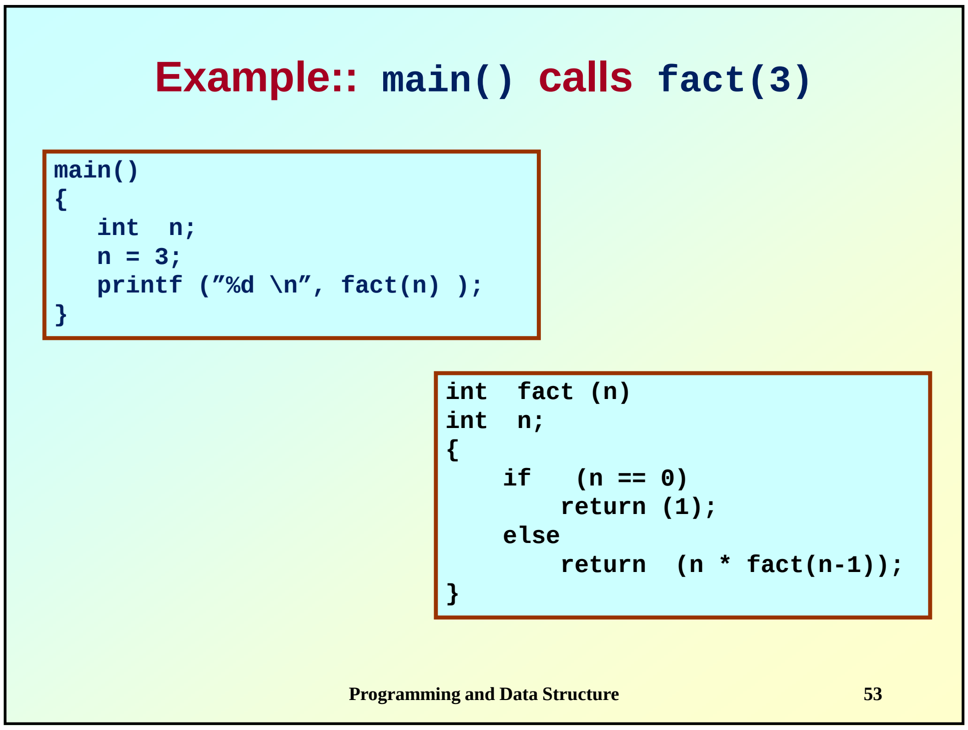
What happens for recursive calls?

- What we have seen ….
  - Activation record gets pushed into the stack when a function call is made.
  - Activation record is popped off the stack when the function returns.
- In recursion, a function calls itself.
  - Several function calls going on, with none of the function calls returning back.
    - Activation records are pushed onto the stack continuously.
    - Large stack space required.
    - Activation records keep popping off, when the termination condition of recursion is reached.



- We shall illustrate the process by an example of computing factorial.
  - Activation record looks like:

Local Variables
Return Value
Return Addr

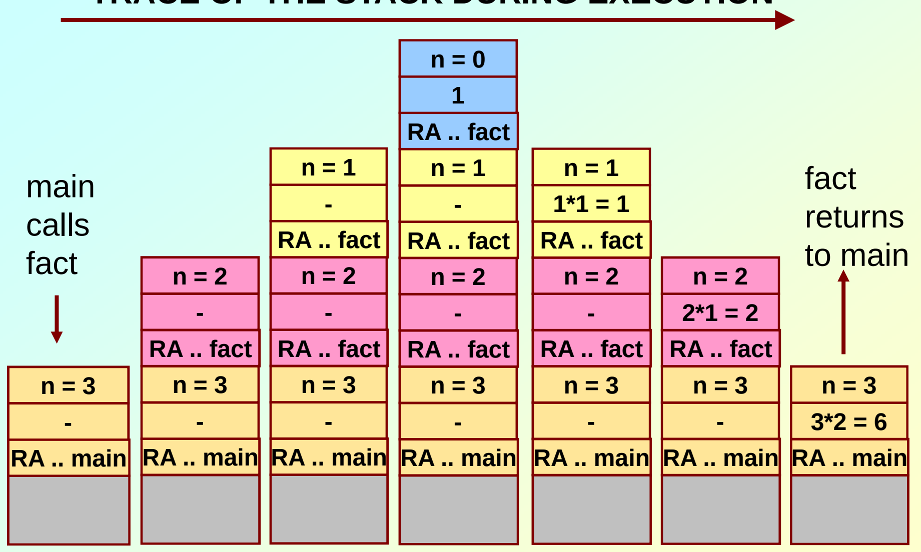


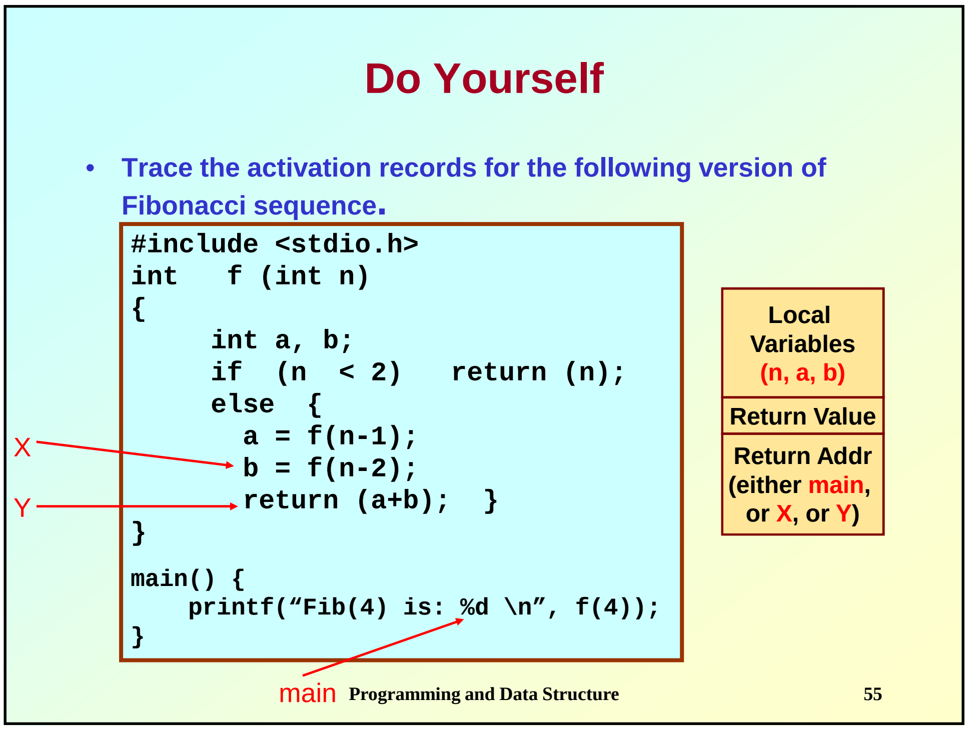
Example:: `main()` calls `fact(3)`

```
main()
{
   int  n;
   n = 3;
   printf (”%d \n”, fact(n) );
}
```

```
int  fact (n)
int  n;
{
    if   (n == 0)
        return (1);
    else
        return  (n * fact(n-1));
}
```

TRACE OF THE STACK DURING EXECUTION



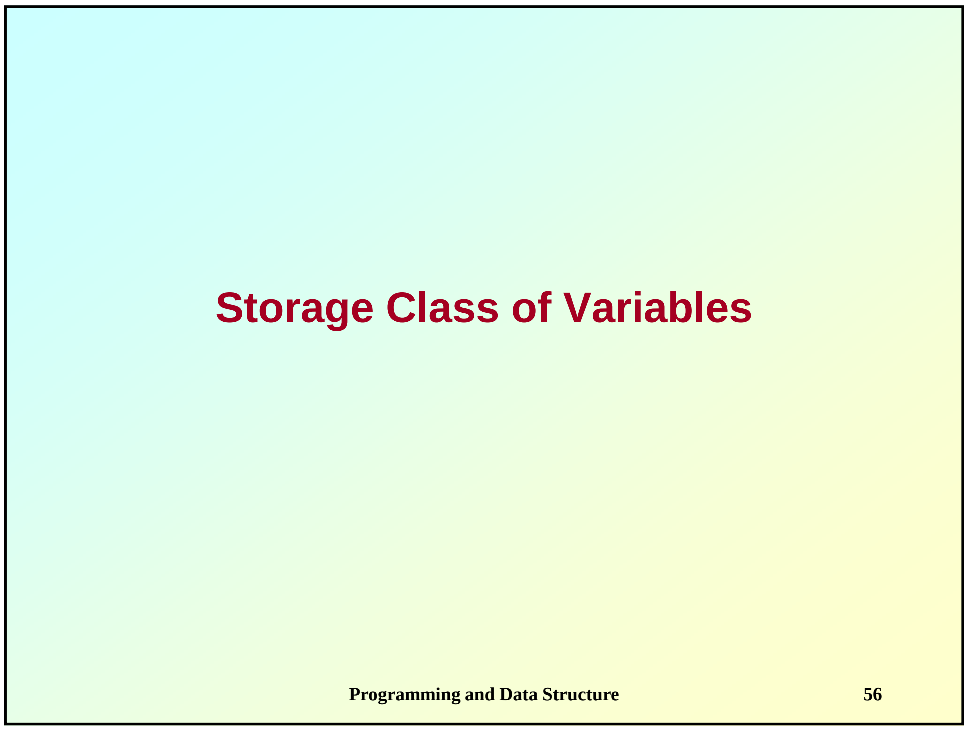


Do Yourself

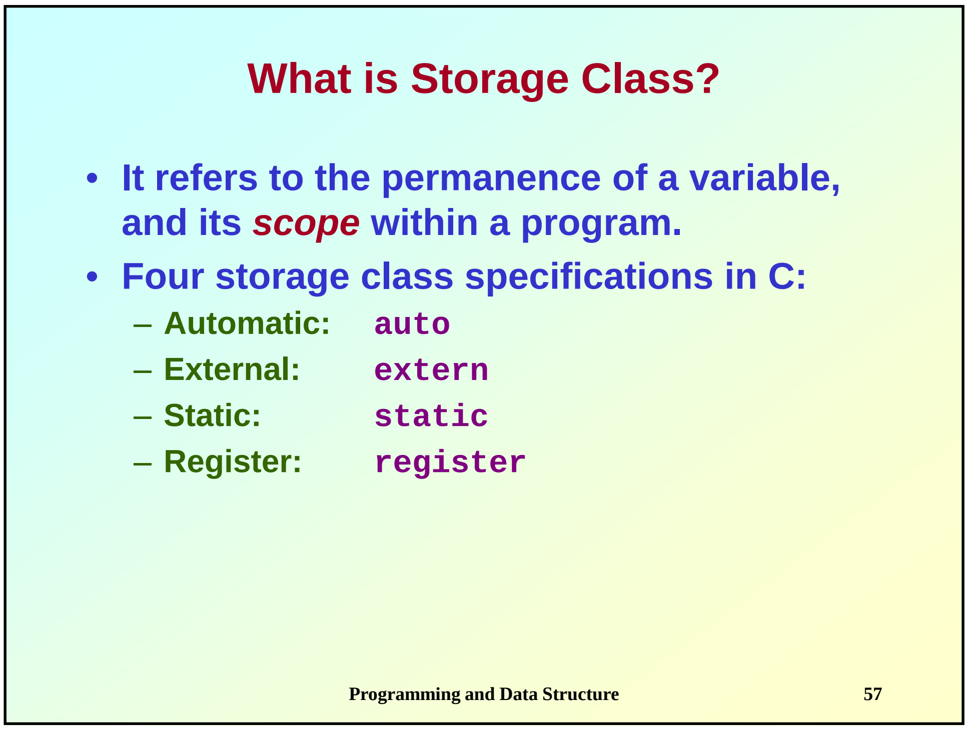
- Trace the activation records for the following version of Fibonacci sequence.

```
#include <stdio.h>
int   f (int n)
{
     int a, b;
     if  (n  < 2)   return (n);
     else  {
       a = f(n-1);
       b = f(n-2);
       return (a+b);  }
}
main() {
    printf("Fib(4) is: %d \n", f(4));
}
```

Local Variables (n, a, b)
Return Value
Return Addr (either main, or X, or Y)



Storage Class of Variables

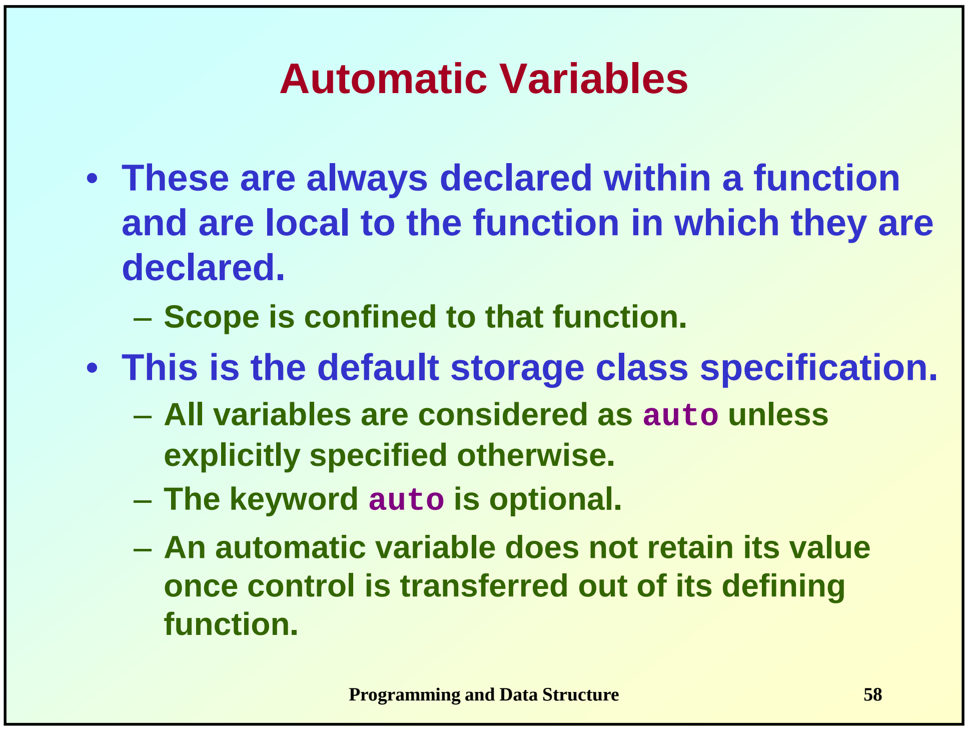


What is Storage Class?

- It refers to the permanence of a variable, and its *scope* within a program.
- Four storage class specifications in C:

– Automatic:	`auto`
– External:	`extern`
– Static:	`static`
– Register:	`register`

Programming and Data Structure 57



Automatic Variables

- These are always declared within a function and are local to the function in which they are declared.
  - Scope is confined to that function.
- This is the default storage class specification.
  - All variables are considered as `auto` unless explicitly specified otherwise.
  - The keyword `auto` is optional.
  - An automatic variable does not retain its value once control is transferred out of its defining function.

Programming and Data Structure 58

```
#include <stdio.h>

int factorial(int m)
{
   auto int i;
   auto int temp=1;
   for (i=1; i<=m; i++)
        temp = temp * i;
   return (temp);
}
```

```
main()
{
   auto int  n;
   for (n=1; n<=10; n++)
     printf (“%d! = %d \n”,
            n, factorial (n));
}
```

Static Variables

- Static variables are defined within individual functions and have the same scope as automatic variables.
- Unlike automatic variables, static variables retain their values throughout the life of the program.
  - If a function is exited and re-entered at a later time, the static variables defined within that function will retain their previous values.
  - Initial values can be included in the static variable declaration.
    - Will be initialized only once.
- An example of using static variable:
  - Count number of times a function is called.

EXAMPLE 1

```
#include <stdio.h>

int factorial (int n)
{
   static int count=0;
  count++;
  printf (“n=%d, count=%d \n”, n, count);
   if (n == 0) return 1;
  else return (n * factorial(n-1));
}
main()
{
    int i=6;
    printf (“Value is: %d \n”, factorial(i));
}
```

- **Program output:**

```
n=6, count=1
n=5, count=2
n=4, count=3
n=3, count=4
n=2, count=5
n=1, count=6
n=0, count=7
Value is: 720
```

EXAMPLE 2

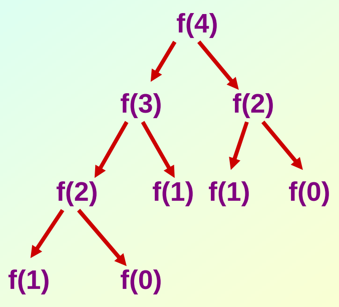
```
#include <stdio.h>

int fib (int n)
{
   static int count=0;
   count++;
   printf (“n=%d, count=%d \n”, n, count);
   if (n < 2) return n;
   else return (fib(n-1) + fib(n-2));
}
```

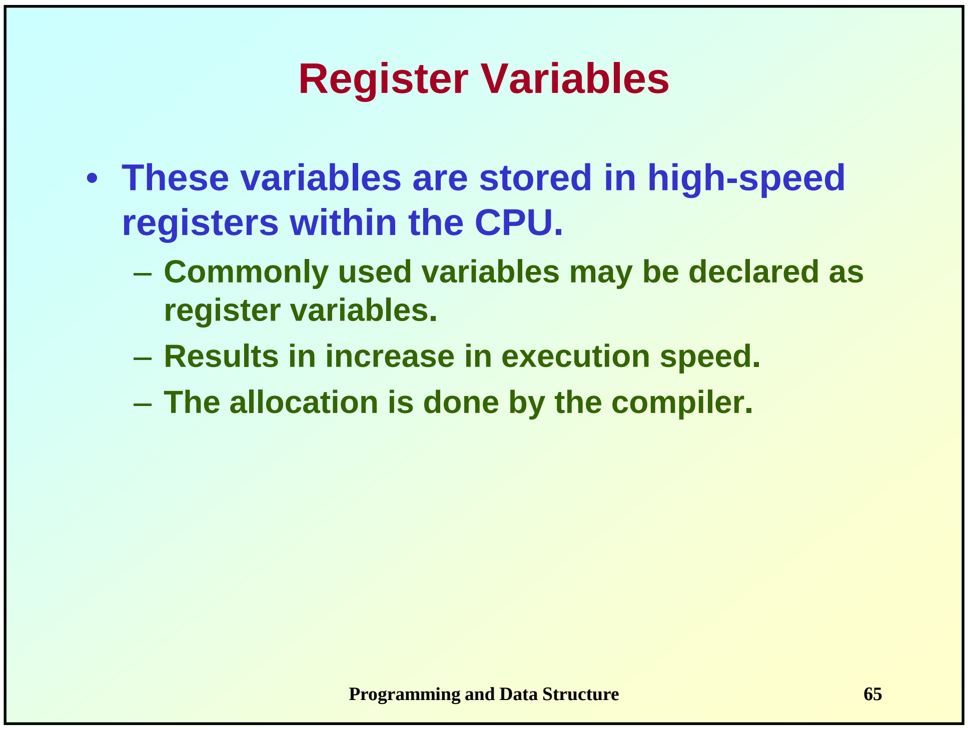
```
main()
{
    int i=4;
    printf (“Value is: %d \n”, fib(i));
}
```

- **Program output:**

```
n=4, count=1
n=3, count=2
n=2, count=3
n=1, count=4
n=0, count=5
n=1, count=6
n=2, count=7
n=1, count=8
n=0, count=9
Value is: 3
```

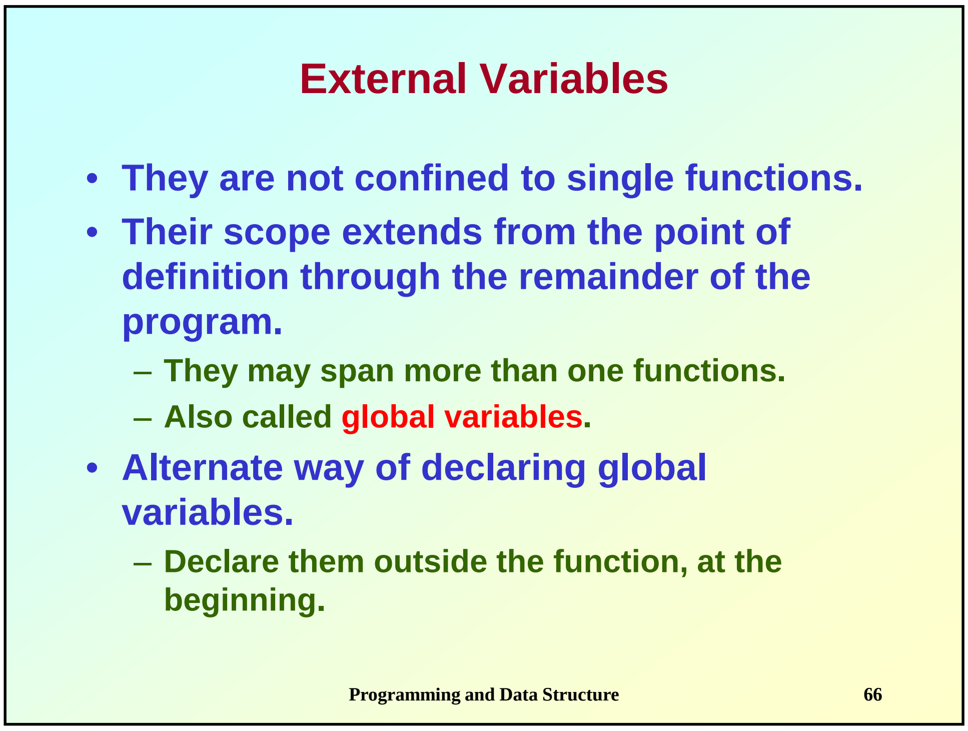


```
[0,1,1,2,3,5,8,….]
```



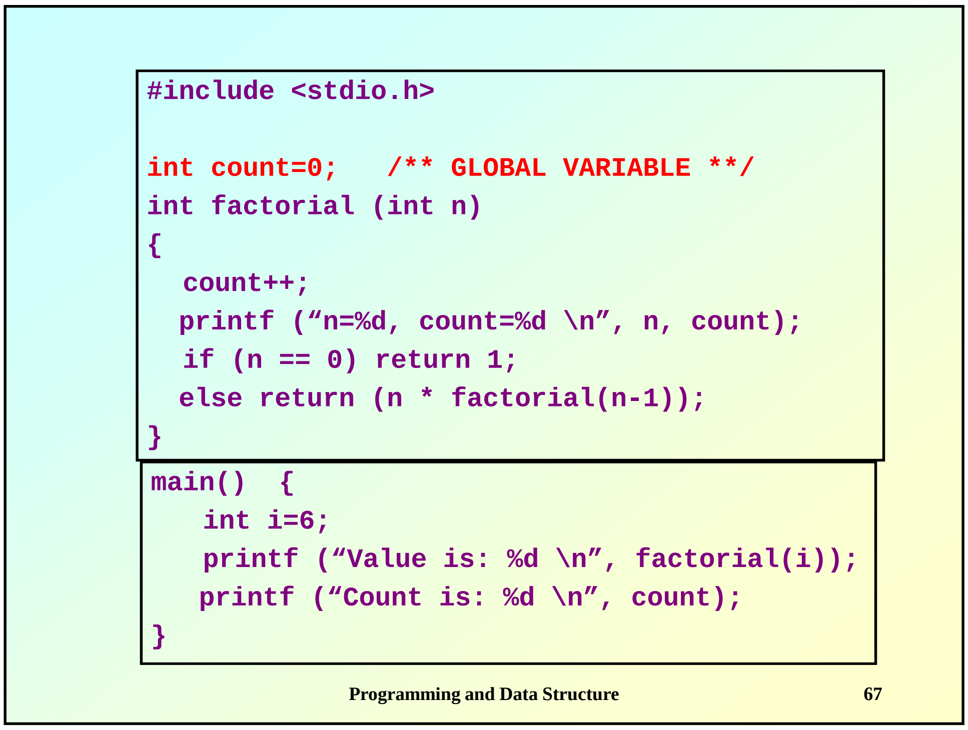
Register Variables

- These variables are stored in high-speed registers within the CPU.
  - Commonly used variables may be declared as register variables.
  - Results in increase in execution speed.
  - The allocation is done by the compiler.



External Variables

- They are not confined to single functions.
- Their scope extends from the point of definition through the remainder of the program.
  - They may span more than one functions.
  - Also called global variables.
- Alternate way of declaring global variables.
  - Declare them outside the function, at the beginning.



```
#include <stdio.h>

int count=0;    /** GLOBAL VARIABLE **/
int factorial (int n)
{
   count++;
  printf (“n=%d, count=%d \n”, n, count);
   if (n == 0) return 1;
  else return (n * factorial(n-1));
}
main()  {
    int i=6;
    printf (“Value is: %d \n”, factorial(i));
   printf (“Count is: %d \n”, count);
}
```