First Year First Semester Course
M.Tech. (CS) [Batch 2021-23]

Lecture #10

Introduction to Programming

Special Topics in C

Course Instructor:

Dr. Monidipa Das

DST-INSPIRE Faculty

Machine Intelligence Unit (MIU), Centre for Artificial Intelligence and Machine Learning (CAIML)

Indian Statistical Institute (ISI) Kolkata, India

Special Topics

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**

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.

auto: Example

```c
#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);
}
```

```c
int main()
{
        auto int n;
        for (n=1; n<=10; n++)
                printf ("%d! = %d \n", n, factorial (n));
        return 0;
}
```

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.

- An example of using static variable:
  – Count number of times a function is called.

static: Example

```c
#include <stdio.h>
void print()
{
  static int count=0;
  printf("Hello World!! ");
  count++;
  printf("is printing %d times.\n",count);
}
int main()
{
  int i=0;
  while(i<10) {
    print();
    i++;
  }
  return 0;
}
```

Output

```
Hello World!! is printing 1 times.
Hello World!! is printing 2 times.
Hello World!! is printing 3 times.
Hello World!! is printing 4 times.
Hello World!! is printing 5 times.
Hello World!! is printing 6 times.
Hello World!! is printing 7 times.
Hello World!! is printing 8 times.
Hello World!! is printing 9 times.
Hello World!! is printing 10 times
```

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.

extern: Example

```
#include <stdio.h>
extern int x = 24;
int b = 6;
int main()
{
   extern int b;
   printf("The value of extern variables x and b : %d,%d\n",x,b);
   x = 15;
   printf("The value of modified extern variable x : %d\n",x);
   return 0;
}
```

extern: More example

file1.h

```
extern int exvar = 9;
```

externprog.c

```
#include "file2.h"
#include <stdio.h>

int main(void)
{
          printf("\nExtern variable value at the beginning:
%d\n",exvar);
    exvar += 11;
    printf("Incremented value: %d\n", exincrem());
    exvar -=6;
    printf("Decremented value: %d\n", exdecrem());
    printf("Extern variable value at the end: %d\n",exvar);
    return 0;
}
```

file3.h

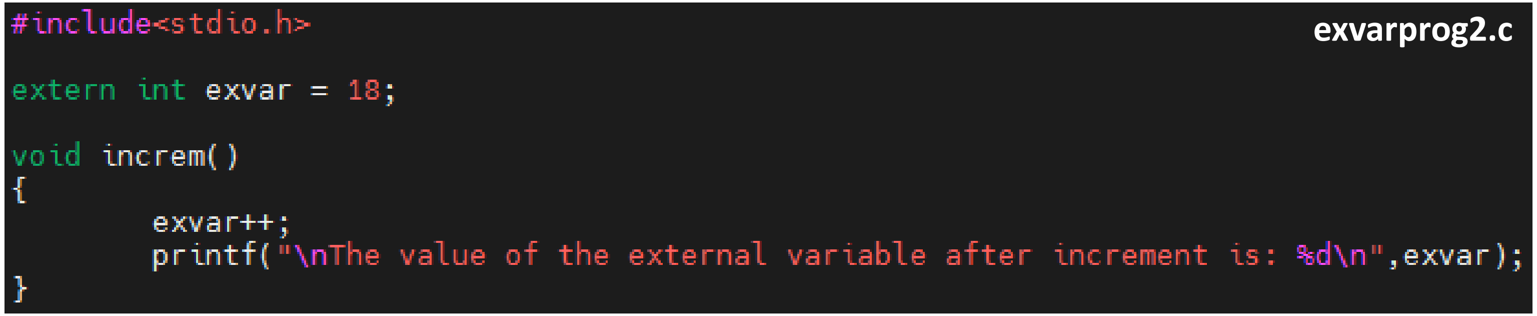
```
int exdecrem(void)
{
    return exvar--;
}
```

file2.h

```
#include "file1.h"
#include "file3.h"

int exincrem(void)
{
    return exvar++;
}
```

extern: More example

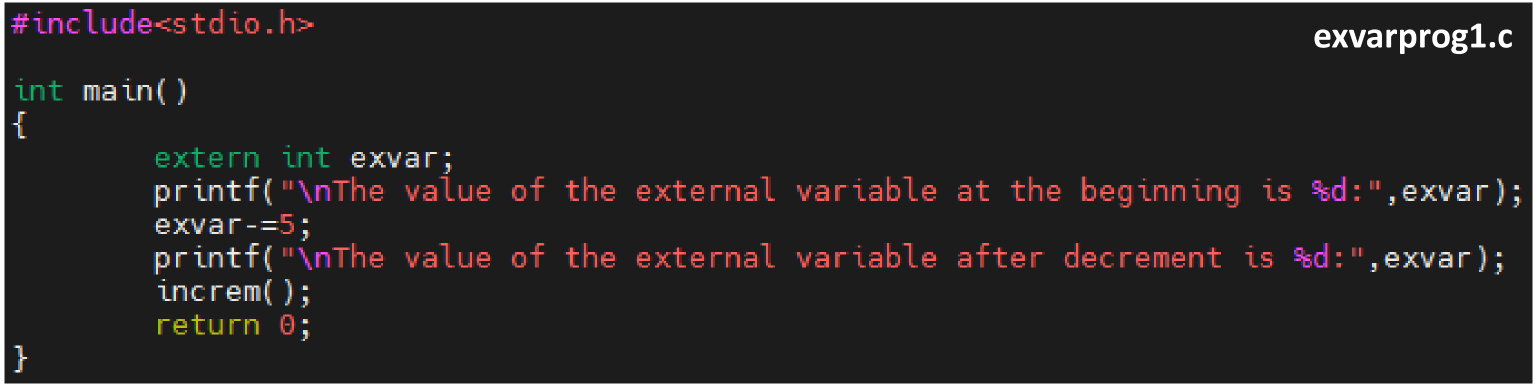


exvarprog2.c

```c
#include<stdio.h>

extern int exvar = 18;

void increm()
{
	exvar++;
	printf("\nThe value of the external variable after increment is: %d\n",exvar);
}
```



exvarprog1.c

```c
#include<stdio.h>

int main()
{
	extern int exvar;
	printf("\nThe value of the external variable at the beginning is %d:",exvar);
	exvar-=5;
	printf("\nThe value of the external variable after decrement is %d:",exvar);
	increm();
	return 0;
}
```

```
Compile: gcc -Wall -o comboprog  exvarprog1.c exvarprog2.c
Execute: ./comboprog
```

global: Example

```
#include <stdio.h>
int count=0;
void print()
{
        printf("Hello World!! ");
        count++;
}
int main()
{
        int i=0;
        while(i<10) {
                print();
                i++;
        printf("is printing %d times.\n",count);
        }
        return 0;
}
```

```
Hello World!! is printing 1 times.
Hello World!! is printing 2 times.
Hello World!! is printing 3 times.
Hello World!! is printing 4 times.
Hello World!! is printing 5 times.
Hello World!! is printing 6 times.
Hello World!! is printing 7 times.
Hello World!! is printing 8 times.
Hello World!! is printing 9 times.
Hello World!! is printing 10 times.
```

static vs global

```
#include <stdio.h>
void print()
{
          static int count=0;
          printf("Hello World!! ");
          count++;
          printf("is printing %d
times.\n",count);
}
int main()
{
          int i=0;
          while(i<10) {
          print();
          i++;
          }
          return 0;
}
```

```
#include <stdio.h>
int count=0;
void print()
{
          printf("Hello World!! ");
          count++;
}
int main()
{
          int i=0;
          while(i<10) {
                    print();
                    i++;
                    printf("is printing %d
times.\n",count);
          }
return 0;
}
```

register Variables

- These variables are stored in high-speed registers within the CPU.
  - Commonly used variables like loop variables/counters may be declared as register variables.
  - Results in increase in execution speed.

```
#include<stdio.h>
int main()
{
      int sum=0;
      register int count;
      for(count=0;count<20;count++)
            sum=sum+count;
      printf("\nSum of Numbers:%d", sum);
      return(0);
}
```

Preprocessor

Preprocessor: Revisited

- A program that processes the source code before it passes through the compiler

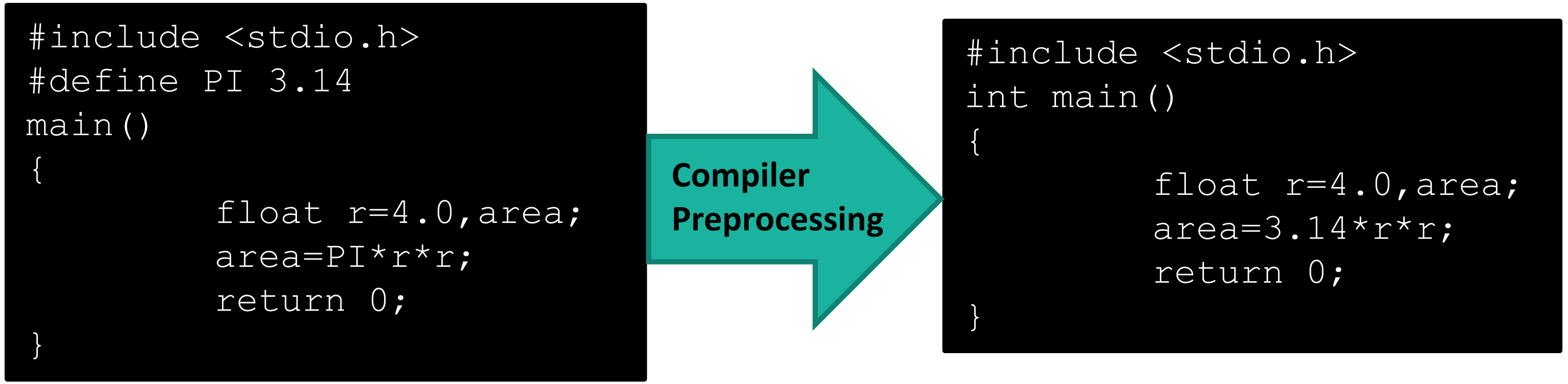
- It operates under the control of ***preprocessor command lines*** or ***preprocessor directives***

- Preprocessor directives follow special syntax rules that are different from the normal C syntax

- Commonly used *preprocessor directives:* **#define, #include**
- Others: #undef, #ifdef, #ifndef, #if, #endif, #else etc.

#define: Macro definition

- Preprocessor directive in the following form *#define identifier string1* replaces the *identifier* by *string1* wherever it occurs before compilation
- e.g.: **`#define PI 3.14`**



```
#include <stdio.h>
#define PI 3.14
main()
{
        float r=4.0,area;
        area=PI*r*r;
        return 0;
}
```

```
#include <stdio.h>
int main()
{
        float r=4.0,area;
        area=3.14*r*r;
        return 0;
}
```

#define with argument

- It may be used with argument

  e.g. **#define sqr(x) ((x)*(x))**

```
#include <stdio.h>
int sqr(int x)
{
          return (x*x);
}
int main()
{
          int y=5;
          printf("value=%d \n", sqr(y)+3);
          return 0;
}
```

```
#include <stdio.h>
int main()
{
int y=5;
printf("value=%d \n", ((y)*(y))+3);
return 0;
}
```

```
#include <stdio.h>
#define sqr(x) ((x)*(x))
int main()
{
      int y=5;
      printf("value=%d \n", sqr(y)+3);
      return 0;
}
```

#define with arguments: A Caution

```
#define sqr(x)  x*x
```

- How macro substitution will be carried out?

```
r = sqr(a) + sqr(30); → r = a*a + 30*30;
r = sqr(a+b); → r = a+b*a+b;
```

WRONG!

- The macro definition should have been written as:

```
#define sqr(x)  ((x)*(x))
```

```
r = ((a+b)*(a+b));
```

#include: File Inclusion

- Preprocessor statement in the following form

```
#include "filename.h"
```

- Filename could be specified with complete path.

```
#include "/home/pralay/C-header/myfile.h"
```

- The content of the corresponding file will be included in the present file before compilation and the compiler will compile thereafter considering the content as it is.

#include: Revisited

```
#include<stdio.h>
int x;

int main()
{
        printf("Give value of x \n");
        scanf("%d",&x);
        printf("Square of x=%d \n",x*x);
}
```

#include “myfile.h”

myprog.c

```
#include <stdio.h>
int x;
```

myfile.h

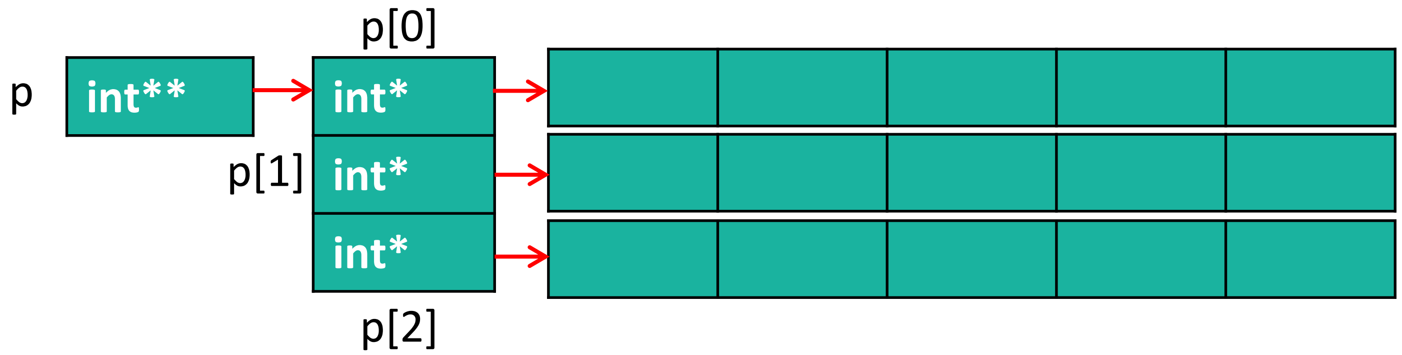
#include<filename.h>

It includes the file “filename.h” from a standard directory.

More on Pointers and Structures

Pointer to Pointer

```
int **p;
p=(int **) malloc(3 * sizeof(int *));
p[0]=(int *) malloc(5 * sizeof(int));
p[1]=(int *) malloc(5 * sizeof(int));
p[2]=(int *) malloc(5 * sizeof(int));
```



Structures within Structures

- Nesting of structures is permitted in C

```
struct salary
{
        char name;

        char department;

        int dearness_ allowance;

        int house_rent_ allowance;

        int city_ allowance;

}employee;
```

↔

```
struct salary
{
        char name;
        char department;
        struct
        {
                int dearness;
                int house_rent;
                int city;

        }allowance;
}employee;
```

Questions?