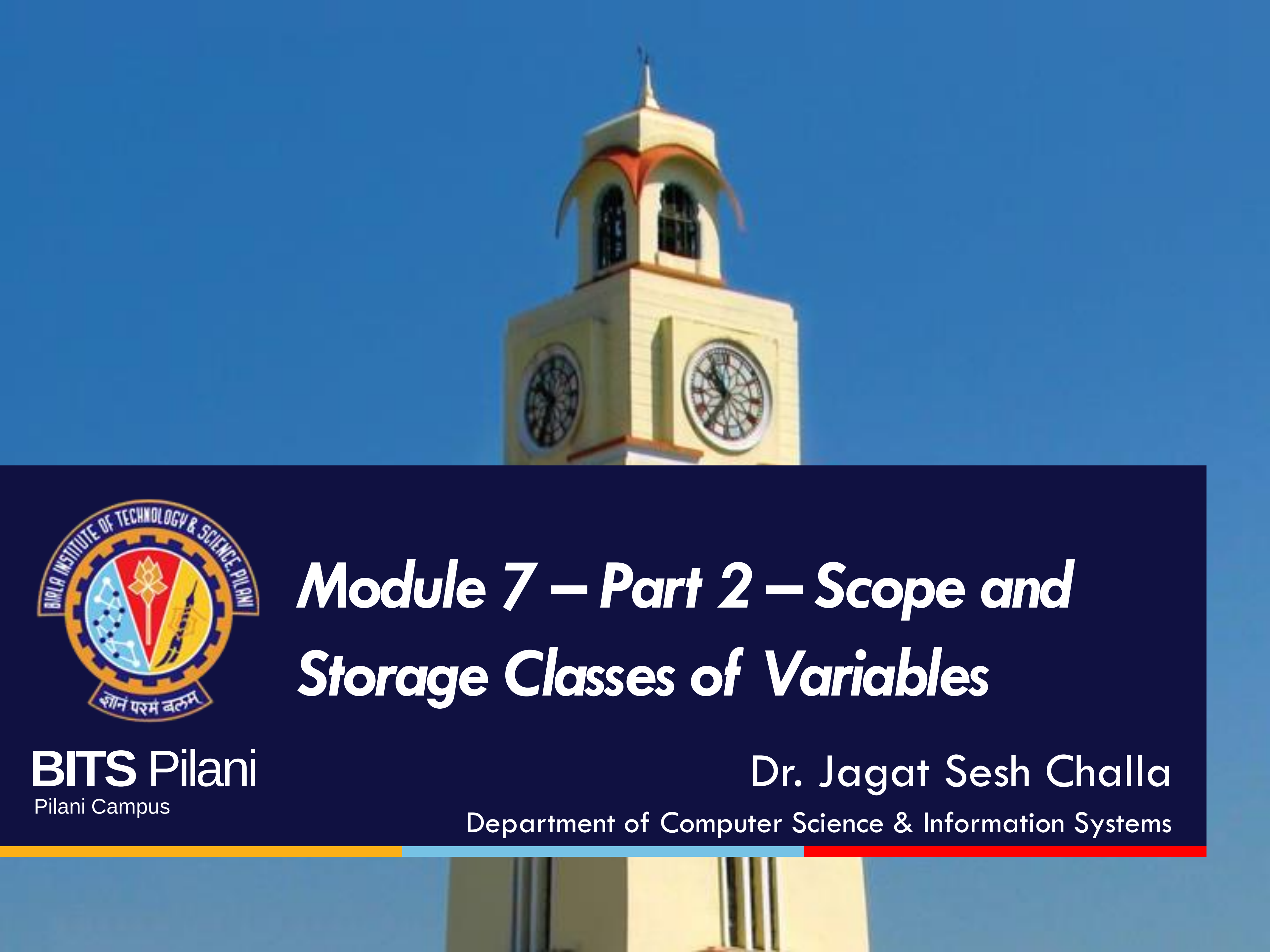


Module 7 – Part 2 – Scope and Storage Classes of Variables

BITS Pilani
Pilani Campus

Dr. Jagat Sesh Challa
Department of Computer Science & Information Systems

Module Overview

- Scope and Storage Class of Variables
- Memory Layout of a C Program
- Auto Variables
- Global Variables
- Static Variables
- Register Variables

Consider this example

```
#include <stdio.h>
void f1(int a){
  printf(“a = %d\n”,a);
  a = 10;
  printf(“a = %d\n”,a);
}
```

```
int main() {
int a = 5;
f1(a);
printf("a = %d",a);
return 0;
}
```

```
output:
a = 5
a = 10
a = 5
```

a is local to the function *f1*

a is local to the function *main*

Scope and Storage Class of a variable

Storage Class: A storage class of a variable tells us about the following:

- the **variable's scope**, or which sections of the code where you can access and use it
- the **location where the variable will be stored** inside the memory
- the **initial value** of a variable
- the **lifetime** of a variable, or how long does the variable reside in the memory

Example - Scope

```
#include <stdio.h>
int y = 5; //scope of y is complete program
void main() {
  int a = 5; //scope of a is main()
  {
     int b = 10; //scope of b is {}
  }
  printf("a = %d, b = %d",a,b); f1();
}
void f1(){
  int x = 2, b = 5; //scope of x and b is f1()
}
```

Scope and Storage Class of a variable

- Auto
- Global
- Static
- Register

innovate achieve lead

BITS Pilani
Pilani Campus

Auto Variables

Auto Storage Class

Variables that are declared within a code block (**{ … }**) are known as **Auto variables** or ***local variables.***

- **Scope: Only the block in which it is declared** can access it
- **Initial Value:** By default, it contains a **garbage value**
- **Storage Location:** Stored in the **stack segment**
- **Lifetime:** Until the execution of the block finishes

```
#include <stdio.h>
void main() {
   int a;
   a = 5;
   {
        int b = 10;
   }
   printf("a = %d, b = %d", a, b);
   f1();
}
void f1(){
   int a;
   a = 20;
}
```

auto variable a local to block of main() function

auto variable b local to the block {}

Error!

auto variable a local to the function f1()

Global Variables

Global Storage Class

Global variables are variables that are **declared outside all the blocks** (or outside all the functions)

- **Scope: Accessible to all the functions** in a program
- **Initial Value:** Default value is **0**
- **Storage Location:** Stored in the **data segment**
- **Lifetime:** Accessible **throughout the program exeucution**

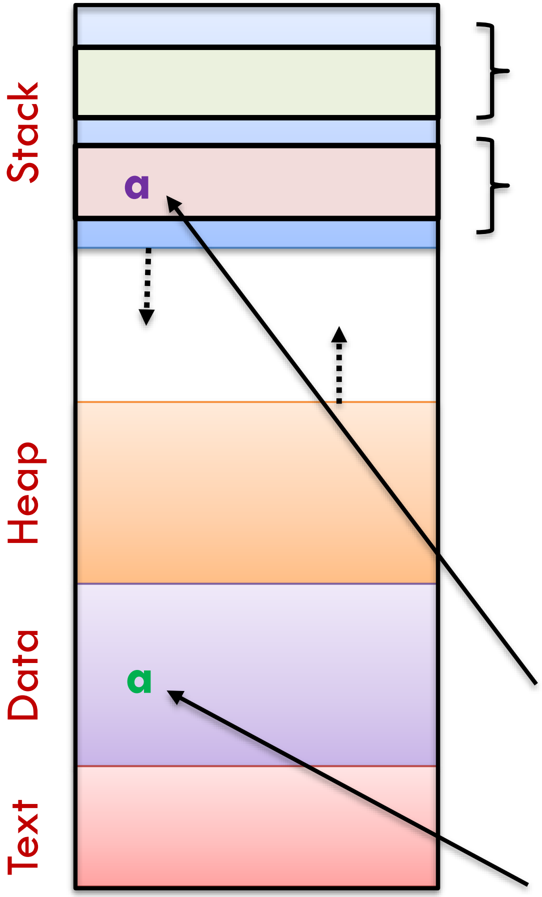
```
#include <stdio.h>
int a;
void f1(){
   printf(“a = %d\n”,++a);
   int a = 2;
   printf(“a = %d\n”, a);
}
void main() {
   a = a+3; f1();
   int a = 5;
   printf("a = %d",a);
}
```

global variable a that is accessible to all functions

Output:

```
a = 4  ←  value of global variable a
a = 2  ←  value of variable a local to f1()
a = 5  ←  value of variable a local to main()
```

Functions, local and global variables in main memory



Frame allocated to *main()* in the stack

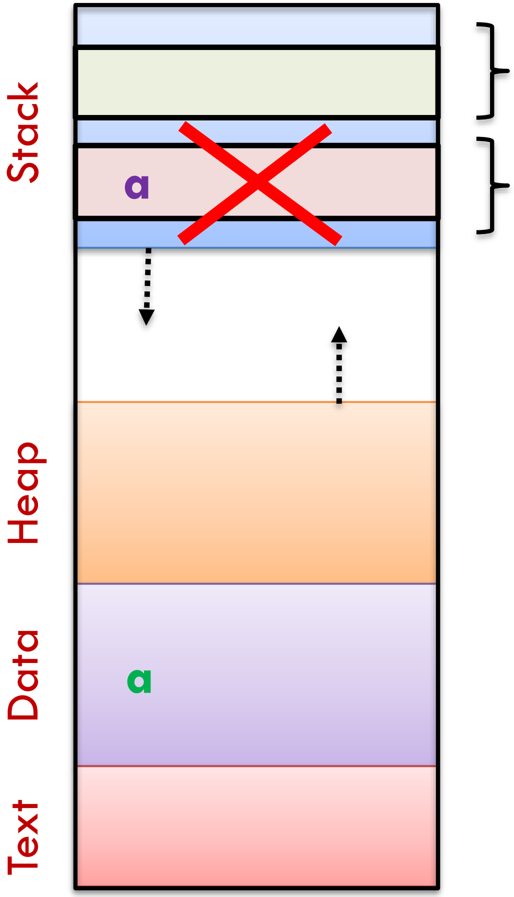
Frame allocated to *f1()* in the stack

local variable a resides in the stack segment, specifically inside the frame allocated to the function f1()

global variable a resides in data segment

```
#include <stdio.h>
int a;
void f1(){
   printf(“a = %d\n”,++a);
   int a = 2;
   printf(“a = %d\n”, a);
}
void main() {
   a = a+3; f1();
   int a = 5;
   printf("a = %d",a);
}
```

Functions, local and global variables in main memory



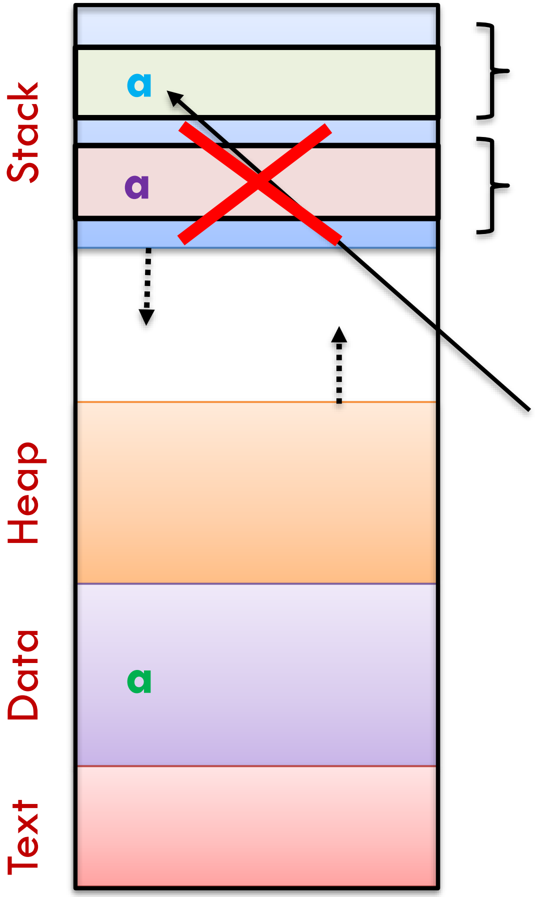
Frame allocated to *main*() in the stack

Frame allocated to *f1*() in the stack

When function *f1()* returns, frame allocated to it is destroyed.

```
#include <stdio.h>
int a;
void f1(){
   printf(“a = %d\n”,++a);
   int a = 2;
   printf(“a = %d\n”, a);
}
void main() {
   a = a+3; f1();
   int a = 5;
   printf("a = %d",a);
}
```

Functions, local and global variables in main memory



Frame allocated to *main*() in the stack

Frame allocated to *f1*() in the stack

local variable a created in the frame allocated to the function main()

When function *f1()* returns, frame allocated to it is destroyed.

```
#include <stdio.h>
int a;
void f1(){
   printf(“a = %d\n”,++a);
   int a = 2;
   printf(“a = %d\n”, a);
}
void main() {
   a = a+3; f1();
   int a = 5;
   printf("a = %d",a);
}
```

Static Variables

Static Variables

Are of two types:

- Local static variable
- Global static variable

Local Static Variables

Static variables are declared using the keyword **static.**

- E.g.- `static int a;`

- **Scope:** Remains **visible only to the function or the block in which it is defined**
- **Initial Value:** Default value is **0**
- **Storage Location:** Stored in the **Data Segment**
- **Lifetime: Until the program terminates**. The value assigned to it remains even after the function (or block) where it is defined terminates
- **Initialized only once**

```
#include <stdio.h>
void main() {
    int i = 0;
    for(i = 0; i < 3; i++)
    {
        static int y = 0;
        y += 10;
        printf(“y = %d\t”,y);
    }
}
```

Local Static Variables

```
#include <stdio.h>
void main() {
int i = 0;
   for(i = 0; i < 3; i++) {
      int y = 0;
      y += 10;
      printf(“y = %d\t”,y);
   }
}
Output: 10 10 10
```

```
#include <stdio.h>
void main() {
int i = 0;
   for(i = 0; i < 3; i++){
      static int y = 0;
      y += 10;
      printf(“y = %d\t”,y);
   }
}
Output: 10 20 30
```

Local static variable

- Retains its value between function calls or block
- Remains visible only to the function or block in which it is defined.
- It remains even after the function terminates

Local Static Variables – Another Example

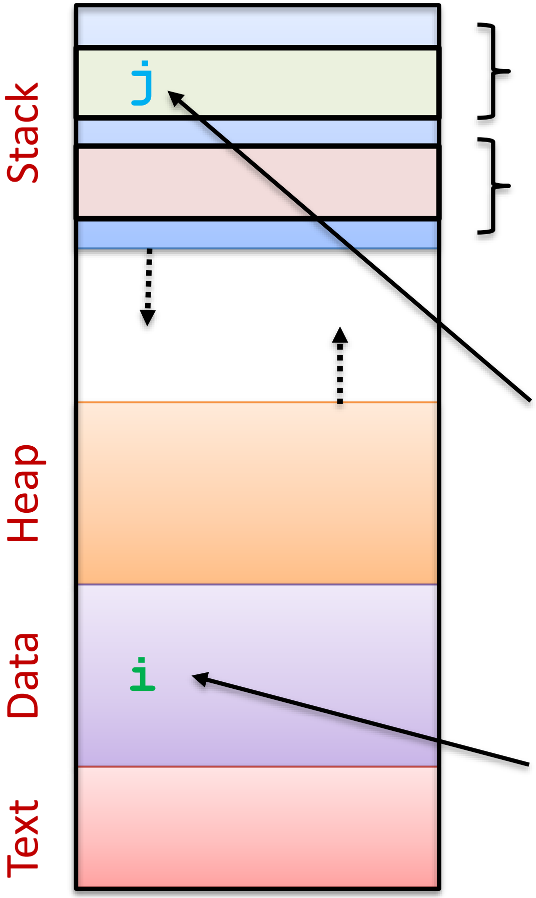
```
#include <stdio.h>
void f1() {
int i = 0;
   i = i+10;
   printf(“i = %d\t”,i);
}
void main() {
int i = 0;
 for(i = 0; i < 3; i++)
      f1();
}
```

Output: 10 10 10

```
#include <stdio.h>
void f1() {
static int i = 0;
   i = i+10;
   printf(“i = %d\t”,i);
}
void main() {
int i = 0;
 for(i = 0; i < 3; i++)
      f1();
}
```

Output: 10 20 30

Local Static Variables in Memory



Stack frame allocated to *main*()

Stack frame allocated to *f1*()

local variable j resides in the stack frame allocated to main()

static variable i resides in data segment even though it is defined inside *f1()*

```
#include <stdio.h>
void f1() {
    static int i = 0;
    i = i+10;
    printf(“i = %d\t”,i);
}
void main() {
    int j = 0;
    for(j = 0; j < 3; j++)
        f1();
}
```

Note: The stack frame for f1() is created and destroyed 3 times. But variable j is created only once and <u>stored in the data segment</u>.

Global Static Variables

- **Scope: Visible to all the functions** in a program
- **Initial Value:** By default, initialized to 0
- **Storage Location:** Stored in the **Data Segment**
- **Lifetime: Until the program terminates**.
- **Initialized only once**
- **Difference with global storage class:** While static global variables are visible only to the file in which it is declared, global variables can be used in other files as well.
  - ***We will study more about multi-file compilation in upcoming lab sessions.***

Example

```
#include <stdio.h>
static int a; //initialized
void f1(){
  printf(“a = %d\n”,a);
  int a = 2;
  printf(“a = %d\n”,a);
}
```

```
void main() {
  f1();
  int a = 5;
  printf("a = %d",a);
}
```

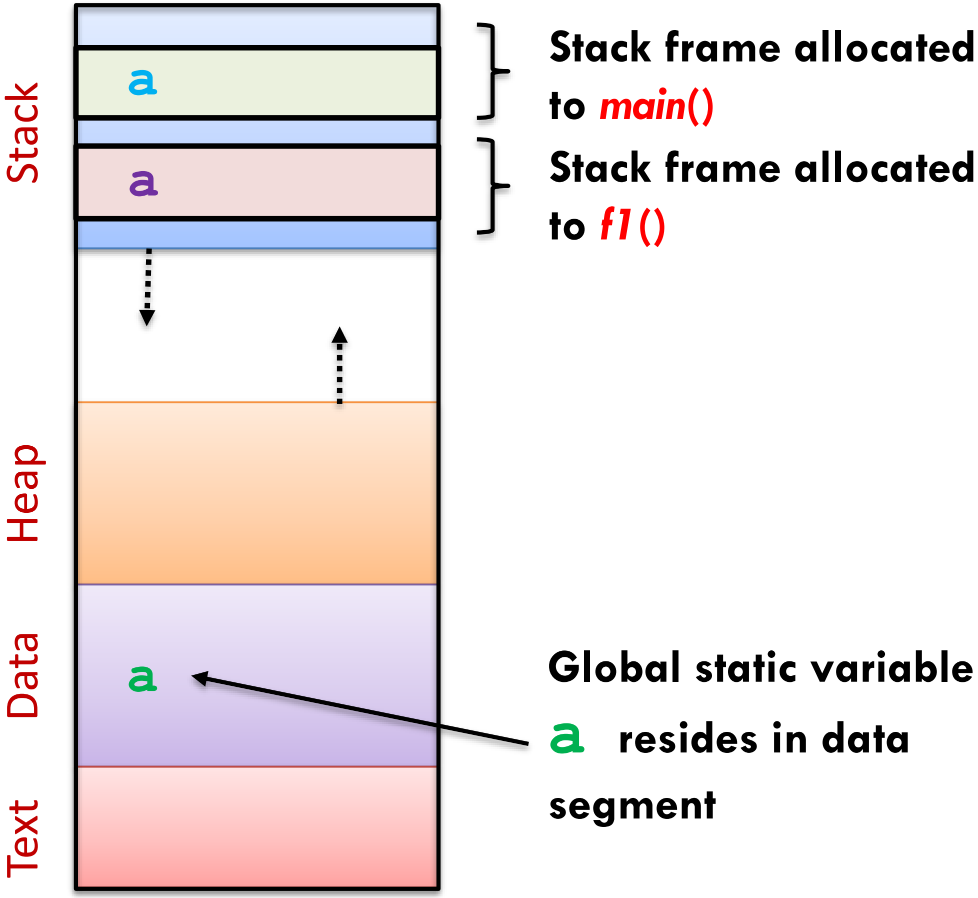
output:

`a = 0` ← *value of static global variable a*

`a = 2` ← *value of auto variable a local to f1*

`a = 5` ← *value of auto variable a local to main*

Memory Allocation of Static global Variable



Stack frame allocated to *main*()

Stack frame allocated to *f1*()

Global static variable a resides in data segment

```
#include <stdio.h>
static int a; //initialized
void f1(){
   printf("a = %d\n",a);
   int a = 2;
   printf("a = %d\n", a);
}
void main() {
   f1();
   int a = 5;
   printf("a = %d",a);
}
```

Register Variables

Registers and Register variables

- A Register is very high-speed memory storage that is located in the CPU.
- Typically limited in number (16 registers for Intel i7 Processor)
- They are extremely fast to access when compared to main memory (RAM).
- The variables that are most frequently accessed can be put into registers using the *register* keyword.
- The keyword *register* hints to compiler that a given variable can be put in a register.
- It's compiler's choice to put it in a register or not.
- Generally, compilers themselves do some optimizations and put the variables in register (even when declared without the register keyword).

Register Variables

- **Scope** – They are local to the function.
- **Default value** – Default initialized value is the garbage value.
- **Lifetime** – Till the end of the execution of the block in which it is defined.

Example:

```
#include <stdio.h>
int main(){
      register char x = ‘S’;    register int a = 10;
      int b = 8;
      printf("The value of register variable b : %c\n",x);
      printf("The sum of auto and register variable: %d",(a+b));
      return 0;
}
```

Output:

```
The value of register variable b : S
The sum of auto and register variable : 18
```

Caveats

- You can't access address of a register variable with an "&"
  - *Can't use register variables with scanf().*
- Register is a storage class, and C doesn't allow multiple storage class specifiers for a variable.
  - *register* can not be used with *static* or *extern.*
- You can't declare global register variables
  - *All register variable must be declared within the functions.*
- There is no limit on number of register variables in a C program.
  - *Compiler may put some variables in register and some not depending upon availability as number of registers is limited.*

Static vs Register Variables

Static Variables	**Register Variables**
Keyword used is – "static".	Keyword used is – "register".
Static variable may be internal or external depending on the place of declaration.	Register variables are declared inside a function.
Local static variables are similar to auto variables or local variables. Whereas global static variables are similar to global variables.	Register variables are similar to auto or local or internal variables.
The execution speed is slower than register variables.	The register variables leads to faster execution of programs.
Internal static variables are active(visibility) in the particular function and External static variables are active in the entire program.	Register variables are active only within the function.
Internal static variables are alive(lifetime) in until the end of the function and External static variables are alive in the entire program.	Register variables are alive until the end of a function.
Static variables stored in initialized data segments.	Register variables are stored in registers.
Static variable is stored in the memory of the data segment.	In register variables, CPU itself stores the data and access quickly.

innovate achieve lead

BITS Pilani
Pilani Campus

Thank you

Q & A