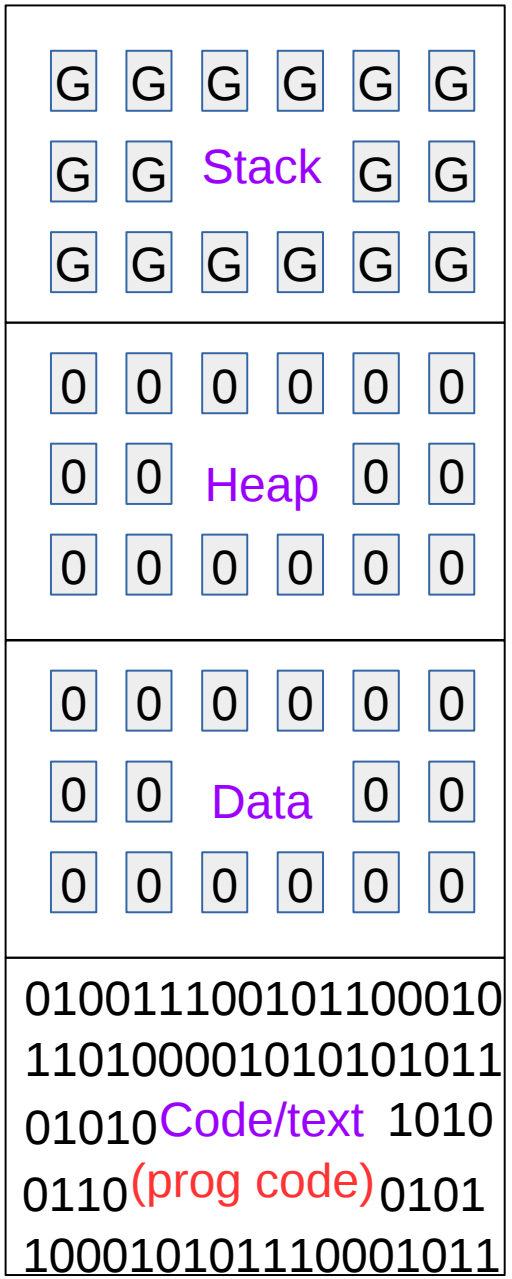
Memory Management



(Runtime Memory)

(Loadtime Memory)

Storage Class 2

For a variable there is not only data type, there is one more attribute also available called storage class.

Syntax : Storage-class Data type variable.

Storage classes provides more information about variables and functions. Storage class provides,

1) Default value.
2) Memory Allocation.
3) Scope (Visibility)
4) Lifetime.

Types of Storage Class

There are four storage classes in C they are as follows:

1) Automatic (auto)
2) Register (register)
3) Static (static)
4) External (extern)

auto

Default value : Garbage value

Memory : Stack

Scope : block/function

Lifetime : block/function

Advantage : As a local variable, provides data security.

Note :

1. The default storage class for local variable is auto
2. All auto variables are local variables, but all local variables are not auto variables.
3. Declaring auto storage class for global variable is not possible.

Example1 :

```
1 #include<stdio.h>
2 int main()
3 {
4         auto int x;
5         printf("x = %d\n",x); // prints Garbage value
6 }
```

Output : x = -1217232896

Example2 :

```
1 #include<stdio.h>
2 auto int x; //compile time error.
3 int main()
4 {
5         printf("x = %d\n",x);
6 }
```

Note : auto variables must be local, should not be global.

Example3 :

```
1 #include<stdio.h>
2 int main()
3 {
4         int x = 10;
5         int x = 20; //re-declaration error.
6         printf("x = %d\n",x);
7 }
```

Note : declaring 2 varibles with same name in same scope is invalid.

Example4 :

```
1 #include<stdio.h>
2 int main()
3 {
4         int x = 10; //life is with in main().
5
6         {
7                 int x = 20; //life is with in block itself.
8                 printf("in block scope : x = %d\n",x); // x = 20
9         }
10
11         printf("in main() scope : x = %d\n",x); // x = 10
12 }
```

Output : in block scope : x = 20

Example5 :

```
1 #include<stdio.h>
2 void fun();
3 int main()
4 {
5         fun();
6         fun();
7         fun();
8 }
9 void fun()
10 {
11         int x = 10;  //re-initialization is done.
12         printf("x = %d\n",x);
13         x++;
14 }
```

Note : auto varibles are re-created if block/function is re-executed.

register

Default value : Garbage value

Memory : Internal CPU registers

Scope : block/function

Lifetime : block/function

Advantage : faster in execution.

Note :

1. Declaring register storage class for global variable is not possible.

2. If there is no enough memory in internal CPU registers, then registers are stored in stack memory.

3. register storage class variables addresses can't be accessed.

Example1 :

```
1 #include<stdio.h>
2 int main()
3 {
4         register int x;
5         printf("x = %d\n",x); // prints Garbage value
6 }
```

Output : x = -1217232896

Example2 :

```
1 #include<stdio.h>
2 register int x; //compile time error.
3 int main()
4 {
5         printf("x = %d\n",x);
6 }
```

Note : register variables must be local, should not be global.

Example3 :

```
1 #include<stdio.h>
2 int main()
3 {
4         register int x = 10;
5         register int x = 20; //re-declaration error.
6         printf("x = %d\n",x);
7 }
```

Note : declaring 2 varibles with same name in same scope is invalid.

Example4 :

```
1 #include<stdio.h>
2 int main()
3 {
4         register int x = 10; //life is with in main().
5
6         {
7                 register int x = 20; //life is with in block itself.
8                 printf("in block scope : x = %d\n",x); // x = 20
9         }
10
11         printf("in main() scope : x = %d\n",x); // x = 10
12 }
```

Output : in block scope : x = 20

Example5 :

```
1 #include<stdio.h>
2 void fun();
3 int main()
4 {
5         fun();
6         fun();
7         fun();
8 }
9 void fun()
10 {
11         register int x = 10;  //re-initialization is done.
12         printf("x = %d\n",x);
13         x++;
14 }
```

Note : register varibles are re-created if block/function is re-executed.

Example6 :

```
1 #include<stdio.h>
2 int main()
3 {
4         register int x;
5         printf("Enter the 'x' value\n");
6         scanf("%d",&x); //compiletime error
7
8         printf("x = %d\n",x);
9 }
```

Note : Accessing addresses of register storage class variables is invalid.

static

Default value : Zero

Memory : Data section

Scope : block/function for local variables
Program for global variables

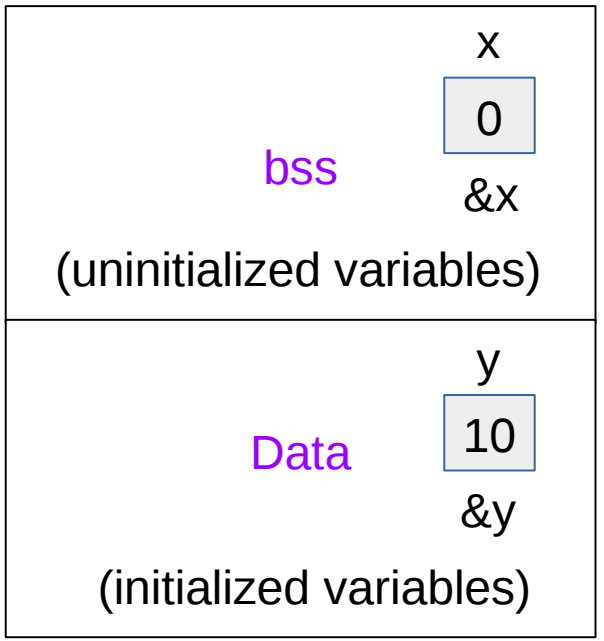
Lifetime : Program

Advantage : static variables are used for counting purpose.

Note :

1. re-initialization is not possible, because the variable can't be re-created.
2. static variables can be local or global.

Data section : It consists of two parts,



Data section

Ex1 : static int x;

Ex2 : static int y = 10;

bss : block started by symbol

Example1 :

```
1 #include<stdio.h>
2 int main()
3 {
4         static int x; //'x' is stored in bss
5         printf("x = %d\n",x);
6 }
```

output : x = 0;

Example2 :

```
1 #include<stdio.h>
2 int main()
3 {
4         static int x = 10;  //'x' stored in data section
5         printf("x = %d\n",x);
6 }
```

output : x = 10;

Example3 :


```
1 #include<stdio.h>
2 static int x = 10;
3 int main()
4 {
5       static int x = 20;
6       printf("x = %d\n",x);
7 }
```

output : x = 20;

Note : If 2 variables having same name as global and as local, then local variable having highest priority in the function.

Example4:

```
1 #include<stdio.h>
2 int main()
3 {
4       static int x = 10; //scope is with in main().
5                          //life is till program execution.
6
7       {
8             static int x = 20; //scope is with in block itself.
9                                //life is till program execution.
10
11             printf("in block scope : x = %d\n",x); // x = 20
12       }
13
14       printf("in main() scope : x = %d\n",x); // x = 10
15 }
```

Example5 :

```
1 #include<stdio.h>
2 void fun();
3 int main()
4 {
5         fun();
6         fun();
7         fun();
8 }
9 void fun()
10 {
11         static int x = 10;  //re-initialization is not done.
12         printf("x = %d\n",x);
13         x++;
14 }
```

Note : static varibles are not re-created, so re-initialization is not done.

Example6 :

```
1 #include<stdio.h>
2 int main()
3 {
4         int x = 10;
5         int y = x;
6         printf("y = %d\n",y);
7 }
```

output : y = 10

Example7 :

```
1 #include<stdio.h>
2 int main()
3 {
4         static int x = 10;
5         static int y = x; //compiletime error
6         printf("y = %d\n",y);
7 }
```

Note :

1. static variables must be initialized with constant value.
2. copying data from one variable to another variable is done at runtime but not at load time.

Strong symbol & Weak symbol :

- 1. for global variables compiler follows the concept of strong symbol and weak symbol concept.
- 2. if global variables are not initialized, then it is known as weak symbol.
- 3. if global variables are initialized, then it is known as strong symbol.
- 4. 2 strong symbols with the same name is not possible.
- 5. 2 weak symbols with the same name is possible.
- 6. 1 strong symbol and 1 weak symbol with same name is possible.

Example6 :

```
1 #include<stdio.h>
2 int main()
3 {
4         int x = 10;
5         int y = x;
6         printf("y = %d\n",y);
7 }
```

output : y = 10

Example7 :

```
1 #include<stdio.h>
2 int main()
3 {
4         static int x = 10;
5         static int y = x; //compiletime error
6         printf("y = %d\n",y);
7 }
```

Note :

1. static variables must be initialized with constant value.
2. copying data from one variable to another variable is done at runtime but not at load time.

Example8:

```
1 #include<stdio.h>
2 int main()
3 {
4       static int x;
5       static int x; //re-declaration error
6       printf("x = %d\n",x);
7 }
```

Example9:

```
1 #include<stdio.h>
2 static int x;
3 static int x; //no error
4 int main()
5 {
6       printf("x = %d\n",x);
7 }
```

output : x = 0;

Example10:

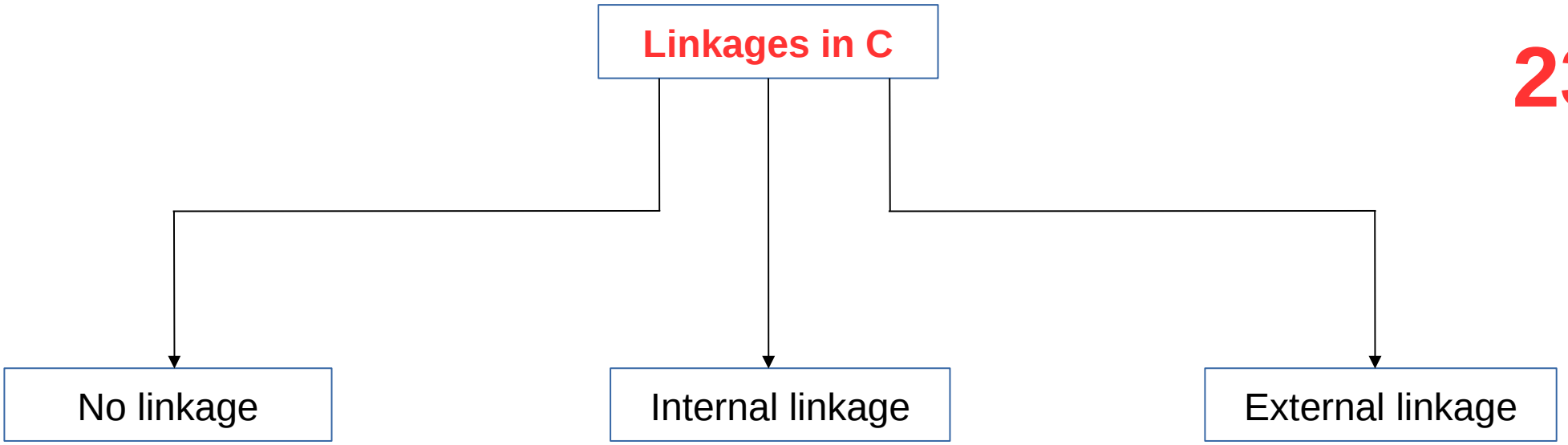
```
1 #include<stdio.h>
2 static int x = 10; //strong symbol
3 static int x; //weak symbol
4 int main()
5 {
6       printf("x = %d\n",x);
7 }
```

output : x = 10;

Example11:

```
1 #include<stdio.h>
2 static int x = 10; //strong symbol
3 static int x = 20; //strong symbol
4 int main()
5 {
6       printf("x = %d\n",x);
7 }
```

output : compiletime error



1. Automatic variables
2. register variables
3. local static variables

1. Global static variables

1. global non-static variables

(extern variables)

No linkage : The identifiers should be used with in the block/function.

Internal linkage : The identifiers should be used with in the same file itself.

External linkage : The identifiers can be used in the same file as well as in other files also, which are belongs to same program.

Multiple file compilation

File 1 (p1.c)

```
#include< stdio.h >
main ( )
{
printf ( “in main function.\n” ) ;
abc ( ) ;
printf ( “After function call.\n” );
printf( “main exit.\n” );
}
```

File 2 (p2.c)

```
void abc ( )
{
printf ( “in abc function.\n” );
}
```

Multiple file compilation

File 1 (p1.c)

cc p1.c

output :

undefined reference to abc()

Note : We can compile more than one file is also possible.

cc p2.c p1.c is also possible

File 2 (p2.c)

cc p1.c p2.c

output :

in main function.
in abc function.
after abc fun call.
main exit.

extern

Default value : Zero

Memory : Data section

Scope : function/program

Lifetime : Program

extern storage class consists of 2 types of statements.

1. extern declaration.
2. extern definition.

→ extern declaration is used to fetch the data from extern definition. extern definition may present in same file or other file of the same program.

→ By default, all global variables are treated as extern storage class variables.

Example1 :

```
1 #include<stdio.h>
2 int x = 10; //extern int x = 10; //extern definition
3 int main()
4 {
5         extern int x; //extern declaration.
6         printf("x = %d\n",x);
7 }
```

Output : x = 10

Example2 :

```
1 #include<stdio.h>
2 int main()
3 {
4         extern int x;  //extern declaration.
5         printf("x = %d\n",x);
6 }
7 int x = 10; //extern definition.
```

Output : x = 10

Example3 :

```
1 #include<stdio.h>
2 int main()
3 {
4         extern int x;  //extern declaration.
5         printf("x = %d\n",x);
6 }
```

Compile : cc prog.c

then it shows linker error, undefined reference to 'x'

Note :

- If an extern declaration variable is accessed in program, it generates linker error if it is unable to find the extern definition.

- Memory is allocated only for extern definition variable, but not for extern declaration variable.

Example4 :

```
1 #include<stdio.h>
2 int main()
3 {
4       extern int x;  //extern declaration.
5       printf("x = %d\n",x);
6 }
```

Prog.c

```
int x = 10; //extern definition
```

file.c

Compile : cc prog.c file.c

Output : x = 10;

Example5:

```
1 #include<stdio.h>
2 int main()
3 {
4       extern int x = 10; //compiletime error
5       printf("x = %d\n",x);
6 }
```

Note : extern keyword variables should not be initialized.
it should be used only for pure declaration purpose.

```
extern int x; // pure declaration.
```

Example6 :

```
1 #include<stdio.h>
2 int main()
3 {
4       extern int x;
5       printf("x = %d\n",x);
6 }
7 static int x = 10; //internal linkage.
```

Compile : cc prog.c
it generates compiletime error.

Note : extern declaration fetches the data only from external linkage definition.

Example7:

```
1 #include<stdio.h>
2 int main()
3 {
4       int x = 10; //no linkage.
5       extern int x;
6       printf("x = %d\n",x); //compiletime error
7 }
```

Compile : cc prog.c
it generates compiletime error.

Example8 :

```
1 #include<stdio.h>
2 int main()
3 {
4       extern int x; //extern declaration.
5       printf("in main(), x = %d\n",x);
6       x = 30;
7       fun();
8 }
```

Prog.c

```
 int x = 10; //extern definition.
1  #include<stdio.h>
2  void fun()
3  {
4       extern int x;
5       printf("in fun(),x = %d\n",x);
6  }
```

file.c

Compile : cc prog.c file.c

Output : in main(), x = 10
in fun(), x = 30

Example9 :

```
1 #include<stdio.h>
2 extern int x;  //strong symbol
3 static int x = 10; //strong symbol
4 int main()
5 {
6         printf("x = %d\n",x);
7 }
```

Compile : cc prog.c
It generates error

Example10:

```
1 #include<stdio.h>
2 static int x = 10; //strong symbol
3 extern int x; //weak symbol
4 int main()
5 {
6         printf("x = %d\n",x);
7 }
```

Output : x = 10;

Example11:

```
1 #include<stdio.h>
2 extern int x;
3 int main()
4 {
5         int x = 10; //local variable
6         printf("x = %d\n",x);
7 }
```

Output : x = 10

Static function : It is function which follows internal linkage, so it must be accessible with in same file scope only.

Note : By default, the function storage class is extern.

```
1 #include<stdio.h>
2 int main()
3 {
4       printf("In main()...\n");
5       abc();
6       printf("after abc()...\n");
7 }
```

Prog.c

```
1 #include<stdio.h>
2 static void abc()
3 {
4       printf("In abc()...\n");
5 }
```

file.c

Compile : cc prog.c file.c ---> compile time error