Module Code: CSE401

Session 4:Control Structure and Storage Class

Session Speaker:

Prema Monish
premamonish.tlll@msruas.ac.in

Session Objectives

- To learn about different types of Control flow statement
- Decision making control structures (Selection statements)
- Iteration or Repetitive or loop control structures (Iteration statements)
- Jump statements
- To learn about different Storage Class Specifier

Session Topics

- if statement
- if-else statement
- nested if statement
- The switch statement
- Go to statement
- While loop
- for loop
- Break
- continue
- do while loop
- Different types Storage class

Control Flow Statements

- All the programs written so far are ***sequential programs***, execute statement after statement in written order from the beginning of main() to the end of main().
- Some times a statement or multiple statements need to be executed, in other time the same thing has to skip from the execution.
- These are classified as
  1. Decision making control structures (Selection statements)
  2. Iteration or Repetitive or loop control structures (Iteration statements)
  3. Jump statements

Decision making control structures

- It is used to select or skip a statement or block of statements according to the value given by the condition.
- These are classified as
  1. If statement
  2. If -else statement
  3. Nested if else statement
  4. else if ladder
  5. Switch-case statement

If statement

- The statement or block of statements is executed if the condition gives true otherwise skipped from the execution.

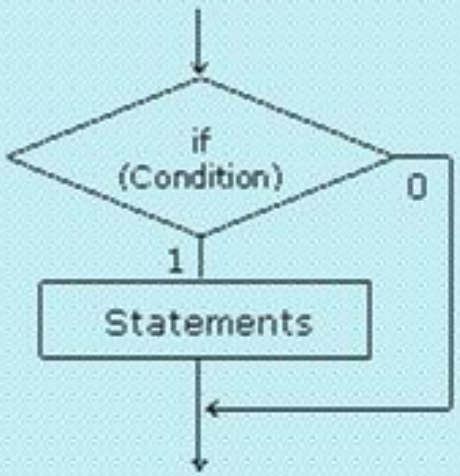
Syntax:

```
if ( Condition )
     statement

     (or)

if ( Condition )
{
    statement
    --------
    --------
}
```

Flowchart



If statement

```
#include<stdio.h>
int main()
{
   printf("Zero");
   if(10<20)
     printf("\nOne");
   if(10>40)
     printf("\nTwo");
   if(1)
     printf("\nThree");
   if(0)
     printf("\nFour");
   printf("\nFive");
   return 0;
}
```

Output:
Zero
One
Three
Five

If -else statement

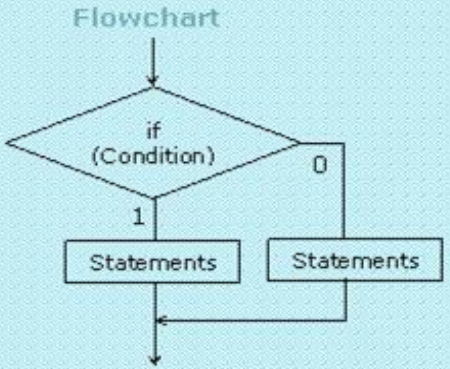
- **if-else** is another selection statement used to solve a problem with only two possibilities by writing a single condition.
- It has two parts that are if part and else part. If the condition is true then the statement or block of statements under the **if** conditional statement is executed otherwise the statement or block of statements under the else part is executed.

Syntax:

```
if ( Condition )
     statement
else
     statement

     (or)

if ( Condition )
{
   statement
   ---------
}
else
{
   statement
   ---------
}
```



If -else statement

```
#include<stdio.h>
int main()
{
    if(10<20)
      printf("One");
    else
      printf("Two");
    printf("\n");
    if(10>20)
      printf("Three");
    else
      printf("Four");
    return 0;
}
```

Output:
One
Four

If –else Algorithms

- Absolute difference between 2 numbers

```
Algorithm absoluteDifference (a, b : Integer)
var temp: Integer;
begin
    if ( a > b ) then
    begin
        temp := a - b;
    end
    else
    begin
        temp := b - a;
    end
    writeln ('The absolute difference is  ', temp);
end
```

Nested if else statement

- Some times we need to check one or more conditions on satisfying or unsatisfying other conditions
- In such case we need to place a selection statement in another selection statement called nesting of selection statements.

Nested if else statement

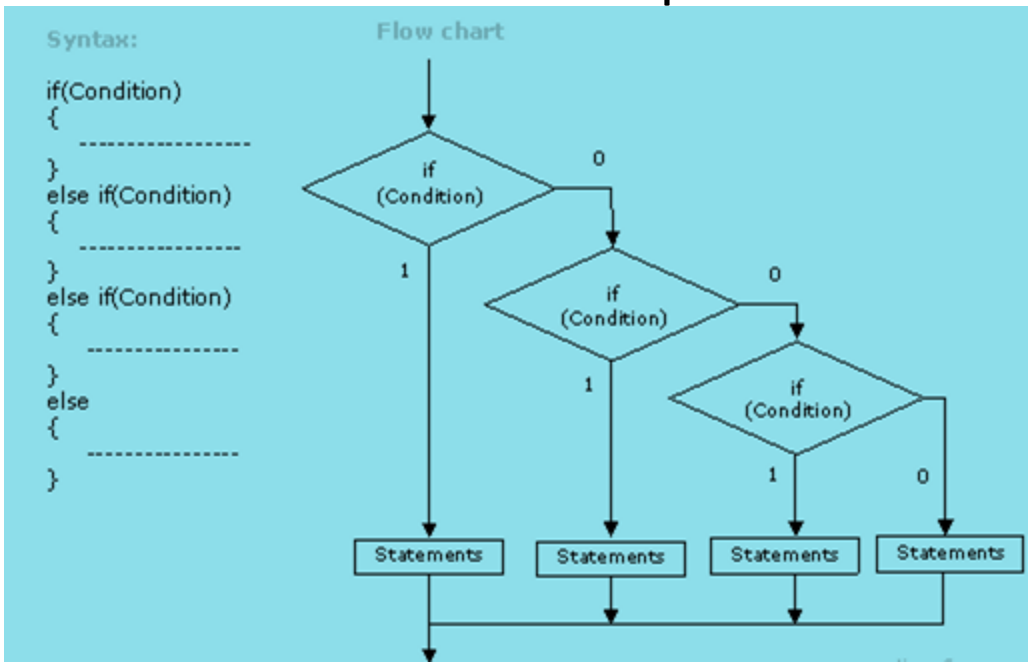
Accept two numbers and print the biggest number.

```
/* using nested if else */
#include<stdio.h>
int main()
{
int a,b;
printf("Enter two integers:\n");
scanf("%d%d",&a,&b);
if(a==b)
  printf("Equals");
else
{
    if(a>b)
      printf("Biggest number %d",a);
    else
      printf("Biggest number %d",b);
}
return 0;
}
```

Execution 1:
Enter two integers:
25 25
Equals
Execution 2:
Enter two numbers:
67 45
Biggest number 67

else if ladder

- else-if ladder is a selection statement used to check multiple conditions, out of which any single or no condition must be true.
- In else-if ladder, condition are checked one by one, if any condition is true then statement(s) under the conditional statement is executed and left all the structure is skipped from the execution.
- If no condition is true then the else part is executed.



else if ladder

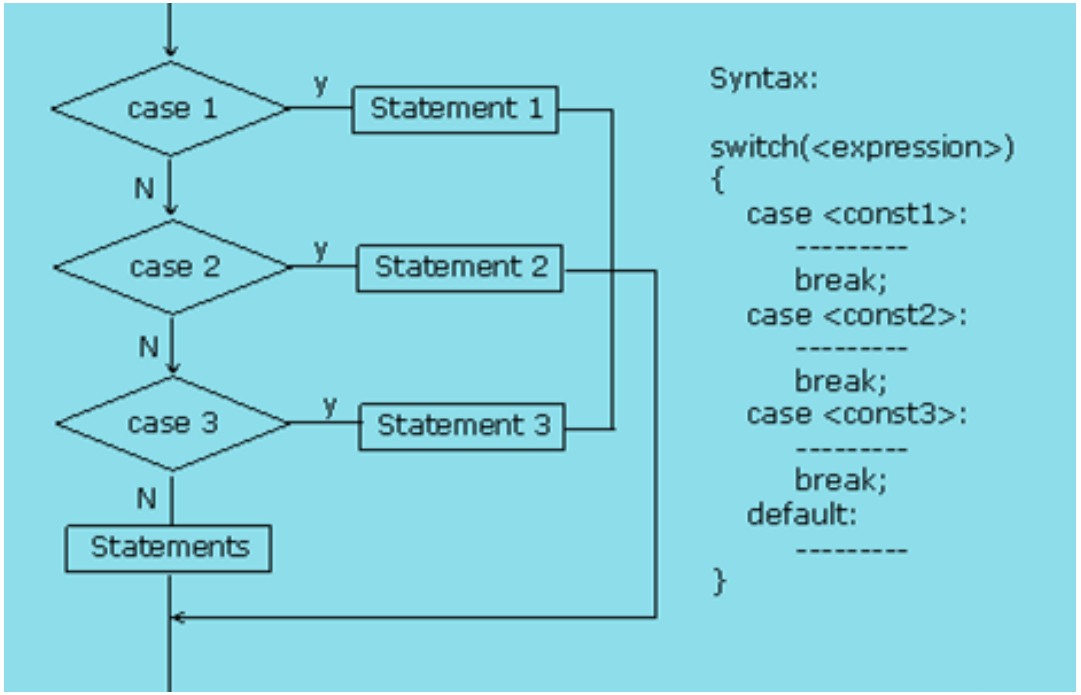
Accept any character from the keyboard and print whether the given character is a vowel, consonant, digit or special symbol.

```
#include<stdio.h>
int main()
{
char ch;
printf("Enter any character:");
scanf("%c",&ch);
if(ch=='a' || ch== 'e' || ch=='i' || ch=='o' || ch=='u')
 printf("Vowel");
else if(ch>='b' && ch<='z') /*checked only when character is not vowel*/
 printf("Consonant");
else if(ch>='0' && ch<='9')/* when character is not an alphabet*/
 printf("Digit");
else
 printf("Special symbol");
return 0;
}
```

switch-case statement

- It is a selection statement used to select a choice from number of choices.
- The switch-case selection statement has the switch statement with an expression and number of case definitions with their constants.
- Only integer and single character constants can be used as case constants.
- If the value of switch expression doesn’t match with any of the case constants then the default case is executed.
- break; statement is used to execute only a single case out of the total structure.

switch-case statement



Syntax:

```
switch(<expression>)
{
   case <const1>:
      ---------
      break;
   case <const2>:
      ---------
      break;
   case <const3>:
      ---------
      break;
   default:
      ---------
}
```

switch-case statement

```
#include<stdio.h>
void main()
{
    int n;
    printf("Enter any integer:");
    scanf("%d",&n);
    switch(n%2)
    {
     case 0:
       printf("Even number");
       break;
     case 1:
       printf("Odd number");
       break;
    }
    return 0;
}
```

Execution 1:
Enter any number: 25
Odd number
Execution 2:
Enter any number: 12
Even number

Iteration or Repetitive or loop control structures

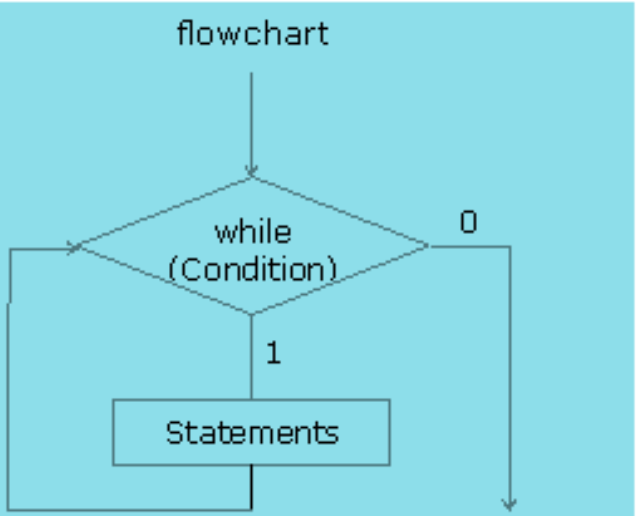
- Executing a statement or multiple statements repeatedly is called a loop or iteration.
- if we want to produce serial numbers from 1 to 20 then we need to print using 20 printf() statements, but it is easier to use a loop to do the same thing.
- This classified into :
  1. While statement
  2. Do-while statement
  3. For statement

While statement

- The while iteration statement has a keyword **while**, a condition to control the loop and a statement or block of statements under the conditional statement.
- If the while condition is false then terminates the execution of loop.

Syntax:

```
while(Condition)
{
    ----------
    ----------
    ----------
}
```



While statement

```
#include<stdio.h>
int main()
{
int x=5;
while(x!=8)
{
   printf("%d\t",x);
   x++;
}
return 0;
}
```

Output:
5 6 7

Algorithm - *while* Statement

while <condition>

begin

…

end

- Example

while (a[i] != n)

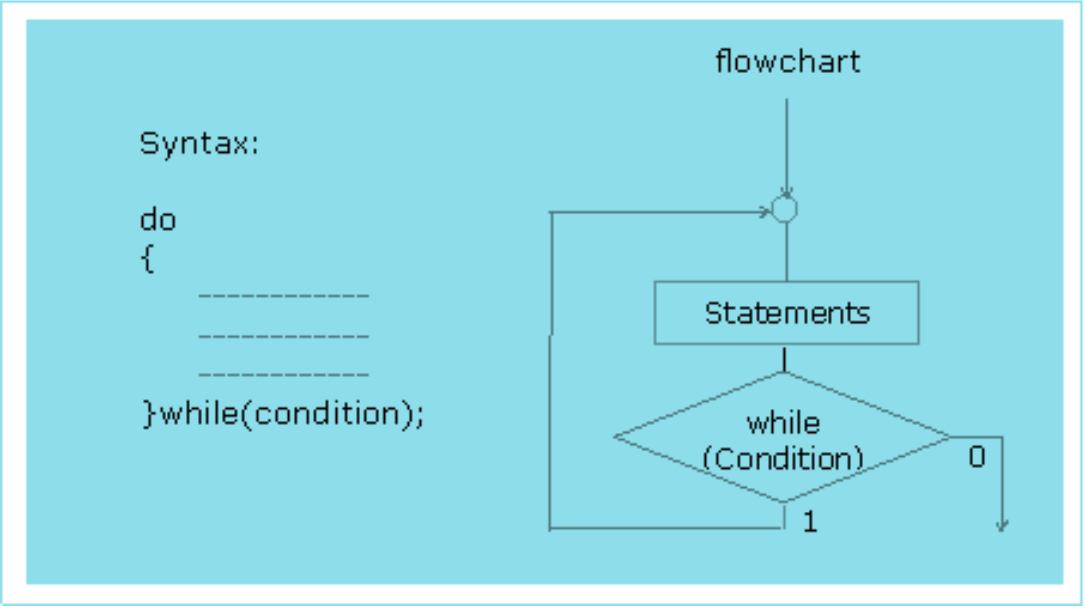
begin

…

end

do-while statement

- do-while is an exit control iteration statement checks the condition after the completion of first iteration and while coming out of the loop.
- If the condition is true then goes to the beginning of the loop to continue iterations otherwise terminates the execution of loop.



do-while statement

- At least one iteration is guaranteed in do-while iteration statement.

```
#include<stdio.h>
int main()
{
  do
  {
    printf("Hello World");
  }while(10>40);/* condition is false */
printf("\nEnd of program..");
return 0;
}
```

Output:
Hello World
End of program..

```
#include<stdio.h>
int main()
{
  while(10>40)
/*condition is false */
  {
    printf("Hello World");
  }
printf("\nEnd of
program..");
return 0;
}
```

Output:
End of program..

Algorithm – *do-while* statement

do

…

while **<condition>**

- Example

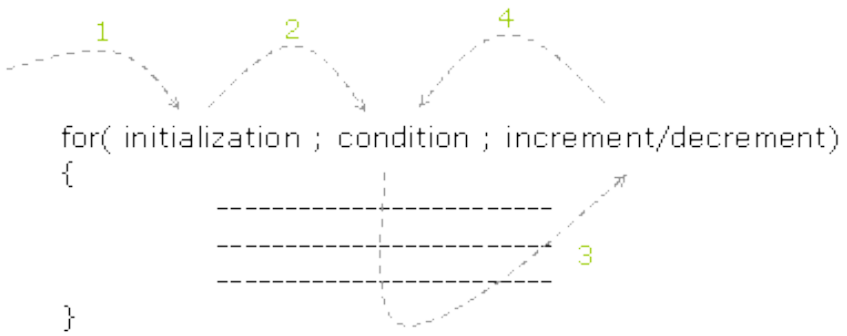
do

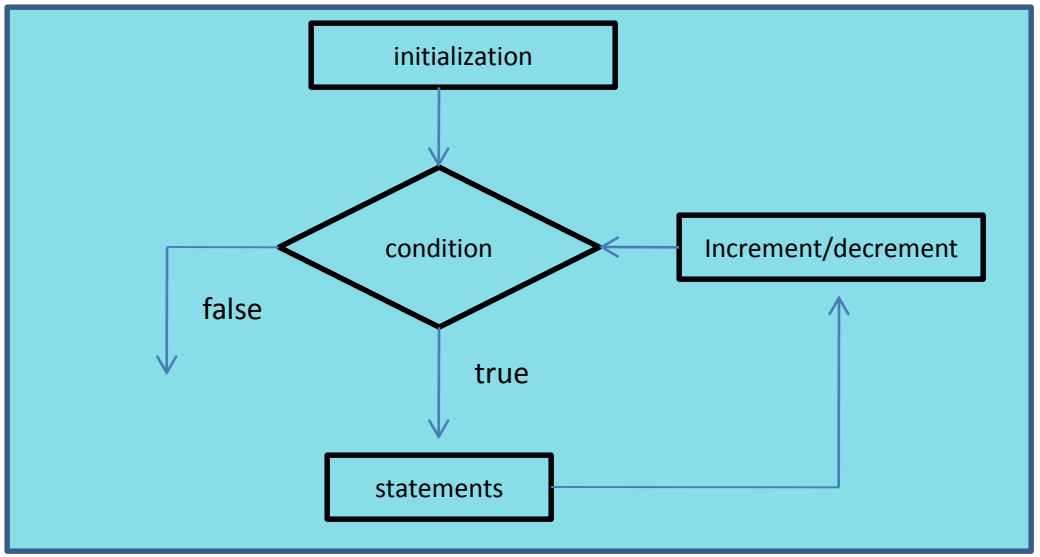
…

while (**a[i] != n)**

For statement

- In case of for loop, three parts that are initialization, condition and increment/decrement statements are integrates in a single statement separated with two semicolons called conditional statement.
- The body is placed just under the for conditional statement.





For statement

- Initialization section is the first section to execute in order to initialize the driver and result variables. It will not be executed again in the total life of the loop.
- Condition is checked, if the condition is true then proceeds to execute the body otherwise exits from the loop.
- The body of the loop is executed and control goes to the increment/decrement section.
- Control goes to the condition and proceeds to the next iterations.

For statement

Printing 1 to 10 natural numbers

```
/* using for loop */
#include<stdio.h>
int main()
{
int i;
for(i=1;i<=10;i++)
  printf("\t%d",i);
return 0;
}
```

```
/* using while loop
*/
#include<stdio.h>
int main()
{
int i=1;
while(i<=10)
{
  printf("\t%d",i);
  i++;
}
return 0;
}
```

For Statement - Algorithms

for <identifier> *in* <initial value> *to* <final value>, *step* <increment> *do*

begin

…

end

- Examples

for i *in* 0 *to* n, *step* 1 *do*

begin

…

end

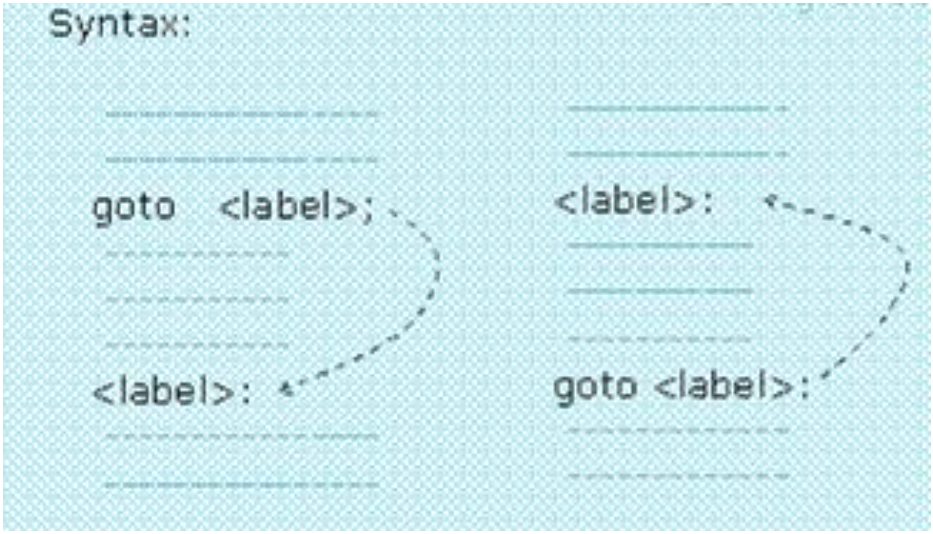
Jump statements

- The purpose of *jump statements* is to send the control wherever you want within the program
- This classified into :
  1. Goto
  2. Break
  3. continue

Goto statement

- The purpose of *goto* is to send the control wherever you want within the program.
- Say for example if we want to send the control from one location to another location then we name the target location with a label and the label is specified with the *goto* keyword.
- So that, the *goto* sends the control to the specified location by skipping the statements in between
- Here the label must be a valid identifier and terminated with a colon (:)

Goto statement



Goto statement

```
#include<stdio.h>
int main()
{
    printf("One");
    printf("\nTwo");
    goto abc;                  /* sends the control to abc: */
     printf("\nThree");
     printf("\nFour");
    abc:
    printf("\nFive");
    return 0;
}
```

Output:
one
Two
Five

Break statement

- The break is also used to jump out of any iteration statement (loop) instantly without going back to test the condition.
- Whenever break encounters with in a loop, control automatically jumps to the first statement after the loop.

continue statement

- it is a jumping statement only used with the iteration statements.
- It is used to jump to the conditional statement of any iteration statement by bypassing the remaining loop.

```
#include<stdio.h>
int main()
{
int x;
for(x=1;x<=10;x++)
{
    if(x%2==1)  /*checking for odd number */
        continue;
    printf("\t%d",x);
}
return 0;
}
```

Output:

2 4 6 8 10

C Errors to Avoid

- Putting = when you mean == in an if or while statement
- Forgetting to increment the counter inside the while loop - If you forget to increment the counter, you get an infinite loop (the loop never ends).
- Accidentally putting a ; at the end of a for loop or if statement so that the statement has no effect - For example:

```
for (x=1; x<10; x++);
printf("%d\n", x);
```

only prints out one value because the semicolon after the for statement acts as the one line the for loop executes.

Storage Class Specifier

- C has four kinds of storage classes
  - Automatic
  - Register
  - Static
  - External
- Storage class is used to denote to the compiler where **memory** is to be allocated for variables
- Storage class tells, what will be the **initial value** of the variable, if the initial value is not specifically assigned.(I.e the default initial value).
- Storage class informs the compiler the **scope** of the variable. Scope refers to which functions the value of the variable would be available.
- Storage class informs the compiler the **life** of the variable. Life refers to how long would the variable exist.

Automatic Storage Class

- All variables declared within a function are called local variables, by default belong to the automatic storage class.

Storage	Stack Memory local to home function
Default Initial Value	Garbage Value
Scope	limited to home function
Life	Till the control remains within the block in which it is defined

Automatic Storage Class

- Auto keyword is the storage class specifier used to declare local variables but, it is optional to write auto while declaring local variables.

```
/* scope of automatic variable */
#include<stdio.h>
int main()
{
 auto int x=10; // int x; is also same meaning
 display();
 return 0;
}
void display()
{
 printf("x=%d",x);
}
```

Output:
Error: undefined symbol “x” in function display()

Register Storage Class

- The register storage class specifier instructs the compiler to store a variable in the registers. A value stored in the register can be accessed faster than one stored in the main memory.
- Global variable can declared as register

Storage	CPU Registers
Default Initial Value	Garbage Value
Scope	Local to the home function
Life	Till the control remains within the block in which it is defined

Register Storage Class

```
main()
{
        register int i;
        for(i=1;i<=10;i++)
                printf("%d\n",i);
/*register variables to improve the overall performance of program*/
}
```

Note: no guarantee that the variable would be stored in register because CPU has limited number of registers and they may be busy in doing some other job. In such case register variables behaves like auto variables.

Static Storage Class

- The keyword static is a storage class specifier used to declare local/internal variables, global/external variables and functions on different purpose

Storage	Data segment/BSS
Default Initial Value	Zero
Scope	Local to the block in which the variable is defined
Life	Value of the variable persists for different function calls

Static Storage Class

```
#include<stdio.h>
void display(int);
int main()
{
     int i;
     for(i=1;i<=5;i++)
      display(i);
     return 0;
}
void display(int x)
{
     int a=100;              /* auto variable */
     static int b=100;      /* static variable */
     a=a+x;
     b=b+x;
     printf("\n%d\t%d",a,b);
}
```

- **b** is a static variable keep its value for the next execution as the life of static variable is permanent
- **a** is an auto variable doesn't keep its value for the next execution

Output:

101	101
102	103
103	106
104	110
105	115

External Storage Class

- Variables declared outside functions i.e. external or global variables have ***external*** storage class
- Extern keyword informs the compiler that, x is an external variable defined some where else in the code and capable to get into the another function.

Storage	BSS and Data segment Memory
Default Initial Value	Zero
Scope	Global
Life	As long as the program's execution doesn't come to an end

External Storage Class

```
/* scope of external/global variables */
#include<stdio.h>
void display();
int main()
{
 printf("x=%d",x);
 display();
 printf("\nx=%d",x);
 return 0;
}
int x=10;//defining external variable
void display()
{
 printf("\nx=%d",x);
 x=x+50;
}
```

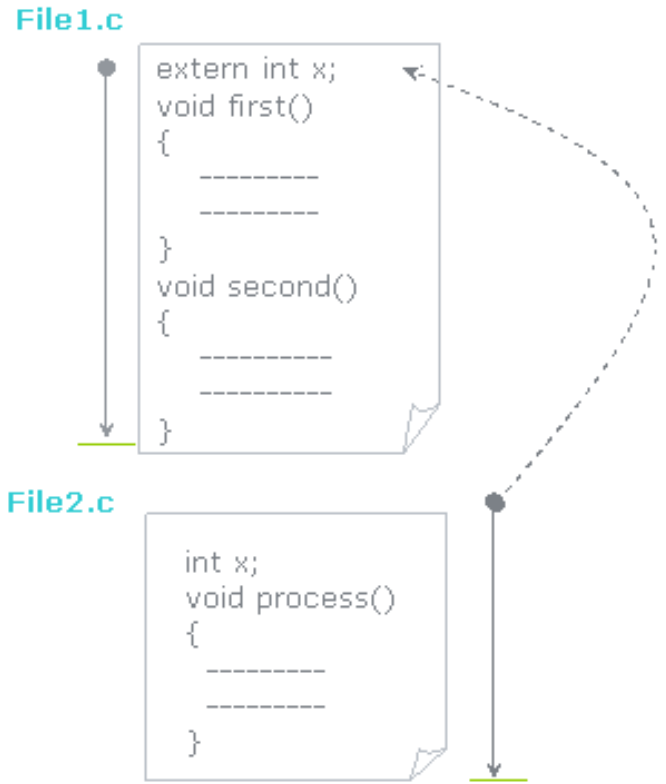
Output:
Error: undefined symbol “x” in function main().

```
/* scope of external variables */
#include<stdio.h>
void display();
int main()
{
 extern int x;
//declaration of extern variable
 printf("x=%d",x);
 display();
 printf("\nx=%d",x);
 return 0;
}
int x=10;//defining external variable
void display()
{
 printf("\nx=%d",x);
 x=x+50;
}
```

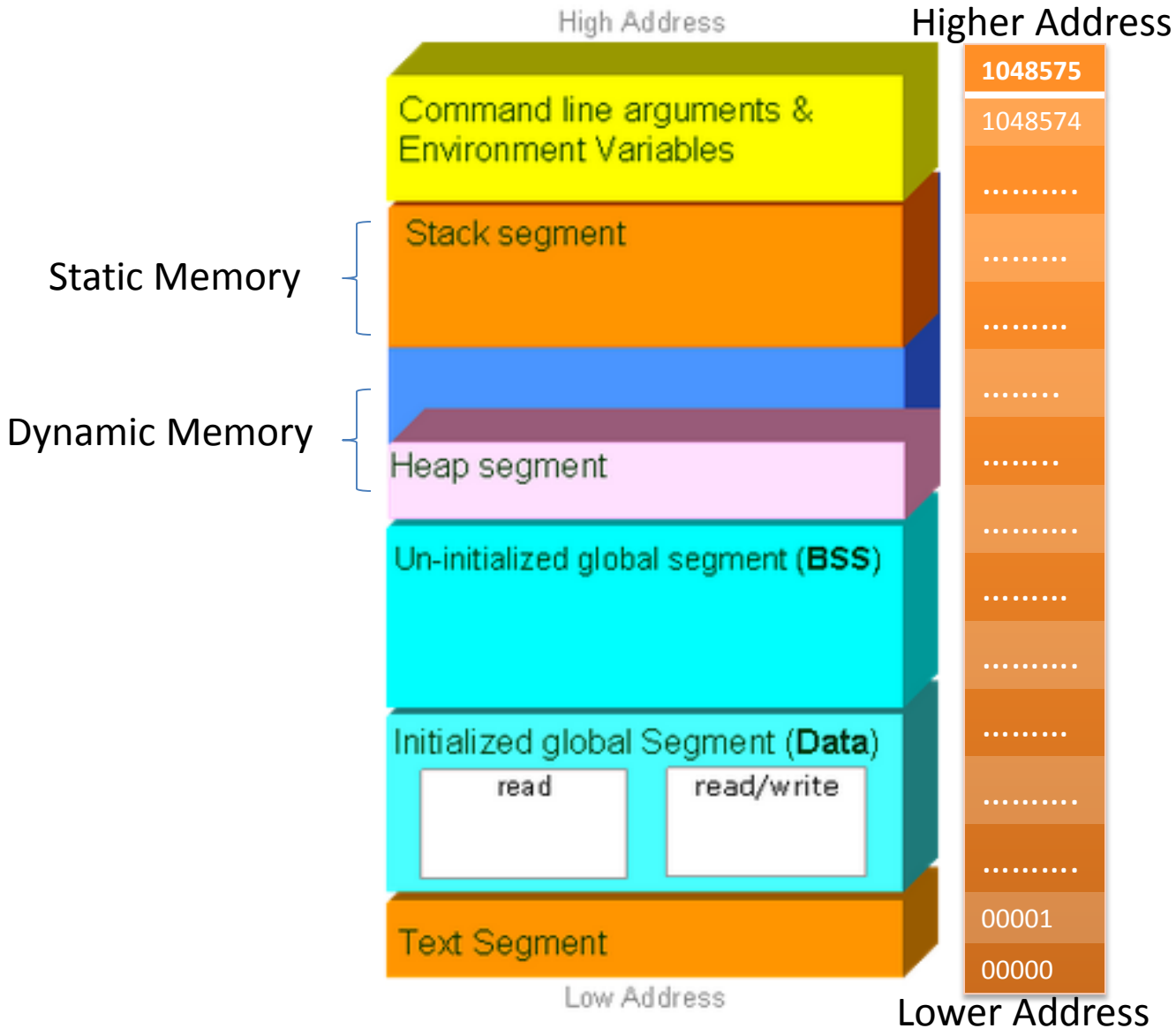
Output:
x=10
x=10
x=60

External Storage Class

- Here x is an external variable defined in file2, can't be accessed into file1 because it's scope is along the file2 only. It can be accessed into file1 by declaring x in file1 using extern storage class specifier.



Memory Segment



Command line arguments

- Interface which allows the user to interact with the computer by providing instructions in the form of typed commands.
- Here parameters are passed to main() function whenever the program is invoked from the command prompt.

```
void main(int argc,char *argv[])
{
……..
……..
}
```

- Where

Argc-integer value means argument count

Argv-array of strings

Command line arguments

```
#include<stdio.h>
void main(int argc,char* argv[])
{
printf("%d  %s",argc,argv[1]);
}
```

```
C:>vim prog.c
C:>cc prog.c
C:>./a.out hello
2  hello
```

```
#include<stdio.h>
void main(int argc,char* argv[])
{
printf("%d  %s",argc,argv[0]);
}
```

```
C:>vim prog.c
C:>cc prog.c
C:>./a.out hello
2  ./a.out
```

Summary

- The different decision structure are if statement, if – else statement, multiple choice else if statement.
- The while loop keeps repeating an action until an associated test returns false.
- The do while loops is similar, but the test occurs after the loop body is executed.
- The for loop is frequently used, usually where the loop will be traversed a fixed number of times.
- Storage class is used to denote to the compiler where memory is to be allocated for variables
- C has four kinds of storage classes
  - Automatic
  - Register
  - Static
  - External

ANY QUERIES