SNS COLLEGE OF TECHNOLOGY

Coimbatore-35
An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF INFORMATION TECHNOLOGY

PROBLEM SOLVING TECHNIQUES AND C PROGRAMMING

I YEAR - I SEM

UNIT 2 – C Programming Basics

TOPIC 5 – Variables

VARIABLES

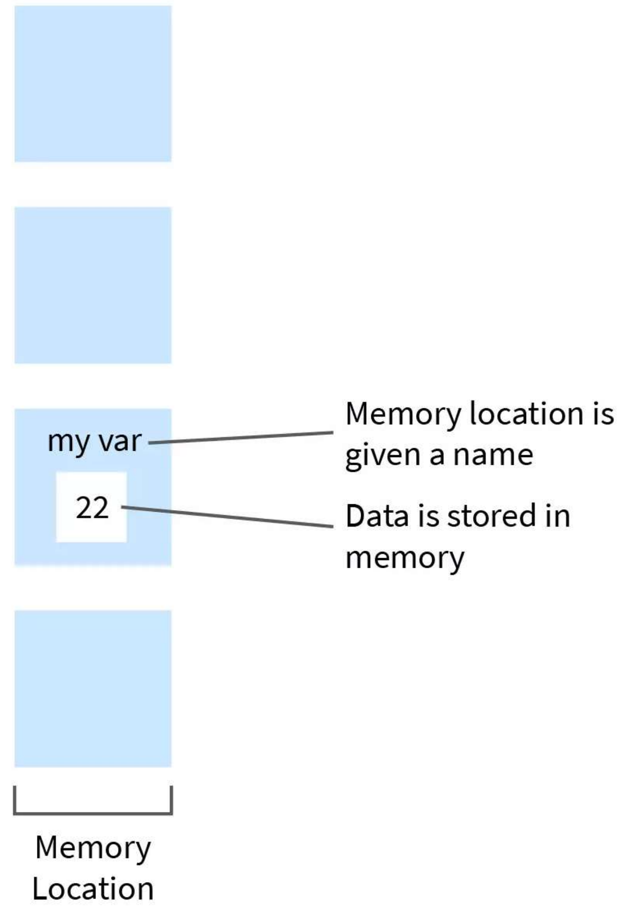
- ✍Variable is basically nothing but the name of a memory location that we use for storing data.
- ✍Variables are the **storage areas** in a code that the program can easily manipulate.
- ✍Every variable in C language has some specific type- that determines the layout and the size of the memory of the variable, the range of values that the memory can hold, and the set of operations that one can perform on that variable.
- ✍The name of a variable can be a composition of digits, letters, and also underscore characters. The name of the character must begin with either an underscore or a letter.
- ✍In the case of C, the lowercase and uppercase letters are distinct. It is because C is case-sensitive in nature.

VARIABLES

- A **variable** is a data name that may be used to store a data value.
- A variable may take different values at different times during execution.
- Some examples of variables' names are:
  - Average
  - height
  - Total
  - Counter_1
  - class_strength
- variable names may consist of letters, digits, and the underscore(_) character

Example:

```
int my_var = 22;
```



RULES FOR NAMING VARIABLES

1. The name of the variable must not begin with a digit.

2. A variable name can consist of digits, alphabets, and even special symbols such as an underscore (_).

3. A variable name must not have any keywords, for instance, float, int, etc.

4. There must be no spaces or blanks in the variable name.

5. The C language treats lowercase and uppercase very differently, as it is case sensitive. Usually, we keep the name of the variable in the lower case.

RULES FOR NAMING VARIABLES

Examples

```
int first_name;      // it is correct
int var1;            // it is correct
int 1var;            // it is incorrect – the name of the variable should not start using a number
int my$var;          // it is incorrect – no special characters should be in the name of the variable
char else;           // there must be no keywords in the name of the variable
int my var;          // it is incorrect – there must be no spaces in the name of the variable
```

DECLARATION OF VARIABLES

Declaration of a variable in a computer programming language is a statement used to specify the variable name and its data type.

Declaration tells the compiler about the existence of an entity in the program and its location.

Syntax

data_type variable_name1, variable_name2, variable_name3;

OR

data_type variable_name;

Example: int radius; char name[50],class; float kilometer;

PRIMARY TYPE DECLARATION

- A variable can be used to store a value of any data type.
  - That is, the name has nothing to do with its type.
- The syntax for declaring a variable is as follows:

  data-type v1,v2,....vn ;

- v1, v2, ...vn are the names of variables.
- Variables are **separated by commas.**
- A declaration statement must end with a semicolon.

- For example, valid declarations are:
  - int count;
  - int number, total;
  - double ratio;

Program to Add Two Integers

```
#include <stdio.h>
int main() {

    int number1, number2, sum;

    printf("Enter two integers: ");
    scanf("%d %d", &number1, &number2);

    // calculating sum
    sum = number1 + number2;

    printf("%d + %d = %d", number1, number2, sum);
    return 0;
}
```

Output

```
Enter two integers: 12
11
12 + 11 = 23
```

- int and double are the keywords to represent integer type and real type data values respectively

PRIMARY TYPE DECLARATION

- A variable can be used to store a value of any data type.
    - That is, the name has nothing to do with its type.
- The syntax for declaring a variable is as follows:

  data-type v1,v2,....vn ;

- v1, v2, ...vn are the names of variables.
- Variables are **separated by commas.**
- A declaration statement must end with a semicolon.

- For example, valid declarations are:
    - int count;
    - int number, total;
    - double ratio;

Program to Add Two Integers

```
#include <stdio.h>
int main() {

    int number1, number2, sum;

    printf("Enter two integers: ");
    scanf("%d %d", &number1, &number2);

    // calculating sum
    sum = number1 + number2;

    printf("%d + %d = %d", number1, number2, sum);
    return 0;
}
```

Output

```
Enter two integers: 12
11
12 + 11 = 23
```

- int and double are the keywords to represent integer type and real type data values respectively

USER-DEFINED TYPE DECLARATION

➢ **typedef Identifier:**

- C supports a feature known as “type definition” that allows **users to ‘define’ an “identifier**” that would represent an existing data type.
- The user-defined data type identifier can later be used to declare variables.
- It takes the general form:
  - » typedef type identifier;
- Where ‘type’ refers to an existing data type and “identifier” refers to the “new” name given to the data type.
- Remember that the new type is ‘new’ only in name, but not the data type.

USER-DEFINED TYPE DECLARATION

- Syntax: typedef type identifier;

- Some examples of type definition are:

```
typedef   int   units;
typedef   float  marks;
```

  - Here, **units** symbolizes **int** and **marks** symbolizes **float**.
- They can be later used to declare variables as follows:

```
units batch1, batch2;
marks name1[50], name2[50];
```

  - Here, batch1 and batch2 are declared as **int** variable and name1[50] and name2[50] are declared as **floating point** array variables.

- The main advantage of typedef is that we can create meaningful data type names for increasing the readability of the program.

USER-DEFINED TYPE DECLARATION

- **enum Identifier:**
- Another user-defined data type is enumerated data type provided by ANSI standard.
- It is defined as follows:

  enum identifi er {value1, value2, ... valuen};

- The "identifier" is a user-defined enumerated data type which can be used to declare variables that can have one of the values enclosed within the braces (known as enumeration constants).
- After this definition, we can declare variables to be of this 'new' type as below:

  enum identifier v1, v2, ... vn;

- The enumerated variables v1, v2, ... vn can only have one of the values value1, value2, ... Value n.

USER-DEFINED TYPE DECLARATION

- Syntax: enum identifier {value1, value2, ... valuen};
- An example:

```
enum day {Monday,Tuesday, ... Sunday};
enum day week_start, week_end;

week_start = Monday;
week_end = Sunday;

if(week_st = = Tuesday)
week_end  = = Monday;
```

- The compiler automatically assigns integer digits beginning with "**0**" to all the enumeration constants.
- That is, the enumeration constant value1 is assigned 0, value2 is assigned 1, and so on.
- However, the automatic assignments can be overridden by assigning values explicitly to the enumeration constants.

DECLARATION OF STORAGE CLASS

- Storage classes in C are used
  - to determine the lifetime,
  - visibility,
  - memory location, and
  - initial value of a variable.

DECLARATION OF STORAGE CLASS

Storage Specifier	Storage	Initial value	Scope	Life
auto	Stack	Garbage	Within block	End of block
extern	Data segment	Zero	global Multiple files	Till end of program
static	Data segment	Zero	Within block	Till end of program
register	CPU Register	Garbage	Within block	End of block

DECLARATION OF STORAGE CLASS - Auto

- Automatic variables are allocated memory automatically at runtime.
- The visibility of the automatic variables is **limited to the block in which they are defined**.
- The scope of the automatic variables is limited to the block in which they are defined. The automatic variables are initialized to garbage by default.
- The memory assigned to automatic variables gets freed upon exiting from the block.
- The keyword used for defining automatic variables is auto.
- Every local variable is automatic in C by default.

DECLARATION OF STORAGE CLASS - Auto

```
≡ File Edit Search Run Compile Debug Project Options Window Help
Help 6
SUM.C 5
[■] AUTO.C 7=[↑]
#include <stdio.h>
#include<conio.h>
int main()
{
int a; //auto
char b;
float c;
clrscr();
printf("%d %c %f",a,b,c); // printing initial default value of automatic var
getch();
return 0;
}
```

```
NeuTroN DOS-C++ 0.77, Cpu speed:
≡ File Edit Search Run
[■]
-28905 ♥ -0.000000_
```

DECLARATION OF STORAGE CLASS - Auto

NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

≡ File Edit Search Run Compile Debug Project Options Window Help

Output

```
11 20 20 20 11
```

LEAP.C

AUTO1.C

```c
#include <stdio.h>
#include <conio.h>S
void main()
{
int a = 10,i;
clrscr();
printf("%d ",++a);
{
int a = 20;
for (i=0;i<3;i++)
{
printf("%d ",a); // 20 will be printed 3 times since it is the local value
}
}
printf("%d ",a); // 11 will be printed since the scope of a = 20 is ended.
getch();
}
```

DECLARATION OF STORAGE CLASS - Static

- The variables defined as static specifier can **hold their value between the multiple function calls.**
- Static local variables are visible only to the function or the block in which they are defined.
- A same static variable can be declared many times but can be assigned at only one time.
- Default initial value of the static integral variable is 0 otherwise null.
- The visibility of the static global variable is limited to the file in which it has declared.
- The keyword used to define static variable is **static.**

DECLARATION OF STORAGE CLASS - Static

NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

≡ File Edit Search Run Compile Debug Project Options Window Help

Output

```
0 0 0.000000 (null)
```

LEAP.C

AUTO1.C

STATIC.C

```c
S#include<stdio.h>
#include<conio.h>
static char c;
static int i;
static float f;
static char s[100];

void main ()
{
clrscr();
printf("%d %d %f %s",c,i,f); // the initial default value of c, i, and f w
getch();
}
```

1:2

F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

DECLARATION OF STORAGE CLASS - Static

NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

≡ File Edit Search Run Compile Debug Project Options Window Help

Output

```
10 24
```

LEAP.C / AUTO1.C / STATIC.C / STATIC1.C

```c
#include<stdio.h>
#include<conio.h>
void sum()
{
static int a = 10;
static int b = 24;
printf("%d %d \n",a,b);
a++;
b++;
}
void main()
{
int i;
clrscr();
for(i = 0; i< 3; i++)
{
```

1:2

Saved to this PC

F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

NeuTroN DOS-C+

≡ File Edit

```
10 24
11 25
12 26
```

DECLARATION OF STORAGE CLASS - Register

- The variables defined as the register is allocated the memory into the CPU registers depending upon the size of the memory remaining in the CPU.
- We can not dereference the register variables, i.e., we can not use &operator for the register variable.
- The access time of the register variables is faster than the automatic variables.
- The initial default value of the register local variables is 0.
- The register keyword is used for the variable which should be stored in the CPU register. However, it is compiler?s choice whether or not; the variables can be stored in the register.
- We can **store pointers into the register, i.e., a register can store the address of a variable.**
- Static variables can not be stored into the register since we can not use more than one storage specifier for the same variable.

DECLARATION OF STORAGE CLASS - Register

NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

≡ File Edit Search Run Compile Debug Project Options Window Help

STATIC1.C

REGISTER.C

```
S#include <stdio.h>
void main()
{
register int b; // variable a is allocated memory in the CPU register. The in
clrscr();
printf("%d",b);
getch();
}
```

DECLARATION OF STORAGE CLASS - External

- The external storage class is used to tell the compiler that the variable defined as extern is declared with an external linkage elsewhere in the program.
- The variables declared as extern are not allocated any memory. It is only declaration and intended to specify that the variable is declared elsewhere in the program.
- The default initial value of external integral type is 0 otherwise null.
- We can only initialize the extern variable globally, i.e., we can not initialize the external variable within any block or method.
- An external variable can be declared many times but can be initialized at only once.
- If a variable is declared as external then the compiler searches for that variable to be initialized somewhere in the program which may be extern or static. If it is not, then the compiler will show an error.

DECLARATION OF STORAGE CLASS - Extern

NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

≡ File Edit Search Run Compile Debug Project Options Window Help

STATIC1.C

REGISTER.C 2

REGISTER.C 6

[■] REGISTER.C [↑]

```c
#include <stdio.h>
int a;
void main()
{
extern int a;
clrscr();
printf("%d",a);
getch();
}
```

DECLARATION OF STORAGE CLASS

- There are four storage class specifiers:

Storage class	Meaning
auto	Local variable known only to the function in which it is declared. *Default is auto*.
static	Local variable which exists and retains its value even after the control is transferred to the calling function.
extern	Global variable known to all functions in the file.
register	Local variable which is stored in the register.

- The storage class is another qualifier (like long or unsigned) that can be added to a variable declaration as shown below:

```
auto int count;
register char ch;
static int x;
extern long total;
```

- Static and external (extern) variables are automatically initialized to zero.
- Automatic (auto) variables contain undefined values (known as 'garbage') unless they are initialized explicitly.

DECLARATION OF STORAGE CLASS

```
#include<stdio.h>
#include<conio.h>
Void main()
{
   int c= 340;
   Printf("C = %d", c);
    {
        int c = 450;
        Printf("C = %d", c);
    }
   Printf("C = %d", c);
   getch();
}
```

Output:
C = 340
C = 450
C = 340

```
#include<stdio.h>
#include<conio.h>
Void main()
{
   int static c= 340;
   Printf("C = %d", c);
    {
        int c = 450;
        Printf("C = %d", c);
    }
   Printf("C = %d", c);
   getch();
}
```

Output:
C = 340
C = 340
C = 340

ASSIGNING VALUES TO VARIABLES

- Variables are created for use in program statements such as:

```
value = amount + inrate * amount;
while (year <= PERIOD)
{
        ....
        ....
        year = year + 1;
}
```

- In the first statement, the **numeric value** stored in the variable **inrate** is multiplied by the value stored in **amount** and the product is added to **amount**.
- The result is stored in the ‘variable’ value.
- This process is possible only if the variables amount and inrate have already been given values.
- The variable value is called the **target variable**.
- While all the variables are declared for their type, the variables that are used in expressions (on the right side of equal (=) sign of a computational statement) must be assigned values before they are encountered in the program.
- Similarly, the variable **year** and the symbolic constant **PERIOD** in the while statement must be assigned values before this statement is encountered.

ASSIGNMENT STATEMENT

- Values can be assigned to variables using the assignment operator "= " as follows:

  variable_name = constant;

- Ex. are:

  initial_value = 0;
  fi nal_value = 100;
  balance = 75.84;
  yes = 'x';

- C permits multiple assignments in one line.
- For example

  initial_value = 0; fi nal_value = 100; are valid statements.

- An assignment statement implies that the value of the variable on the **left** of the 'equal sign' is set equal to the value of the quantity (or the expression) on the **right**.
- The statement:

  year = year + 1;

  - means that the 'new value' of year is equal to the 'old value' of year plus 1.

ASSIGNMENT STATEMENT

- During assignment operation, C converts the type of value on the right-hand side to the type on the left.
- This may involve **truncation** when real value is converted to an integer.
- It is also possible to assign a value to a variable at the time the variable is declared.
- This takes the following form:

  data-type variable_name = constant;
- Some examples are:

```
int fi nal_value = 100;
char yes = 'x';
double balance = 75.84;
```

- The process of giving initial values to variables is called **initialization**.
- C permits the initialization of more than one variables in one statement using multiple assignment operators.
- For example

```
p = q = s = 0;
x = y = z = 10;
```

- are valid. The first statement initializes the variables p, q, and s to zero while the second initializes x, y, and z with 10.

READING DATA FROM KEYBOARD

- Another way of giving values to variables is to input data through keyboard using the **scanf** function.
- It is a general input function available in C and is very similar in concept to the **printf** function.
- It works much like an INPUT statement.
- The general format of **scanf** is as follows:

  scanf(“control string”, &variable1,&variable2,....);

- The control string contains the format of data being received.
- The ampersand symbol **&** before each variable name is an operator that specifies the variable name’s address.

```
#include <stdio.h>
int main()
{
int number1, number2, sum;
printf("Enter two integers: ");
scanf("%d %d", &number1, &number2);
sum = number1 + number2;
printf("%d + %d = %d", number1, number2, sum);
}
```

OUTPUT:

```
 Enter two integers:  12    11
12+11 = 23
```

READING DATA FROM KEYBOARD

```
#include <stdio.h>
int main()
{
int number1, number2, sum;
printf("Enter two integers: ");
scanf("%d %d", &number1, &number2);
sum = number1 + number2;
printf("%d + %d = %d", number1, number2, sum);
}
```

OUTPUT:

```
Enter two integers:  12    11
12+11 = 23
```

```
scanf("%d %d", &number1, &number2);
```

- When this statement is encountered by the computer, the execution stops and waits for the value of the variable number to be typed in.
- Since the control string “%d” specifies that an integer value is to be read from the terminal, we have to type in the value in integer form.
- Once the number is typed in and the ‘Return’ Key is pressed, the computer then proceeds to the next statement.
- Thus, the use of scanf provides an interactive feature and makes the program ‘user friendly’.

READING DATA FROM KEYBOARD

Entire Data types in c:

Data type	Size(bytes)	Range	Format string
Char	1	128 to 127	%c
Unsigned char	1	0 to 255	%c
Short or int	2	-32,768 to 32,767	%i or %d
Unsigned int	2	0 to 65535	%u
Long	4	-2147483648 to 2147483647	%ld
Unsigned long	4	0 to 4294967295	%lu
Float	4	3.4 e-38 to 3.4 e+38	%f or %g
Double	8	1.7 e-308 to 1.7 e+308	%lf
Long Double	10	3.4 e-4932 to 1.1 e+4932	%lf

DEFINING SYMBOLIC CONSTANTS

#define symbolic-name value of constant

- Valid examples of constant definitions are:
  - #define STRENGTH 100
  - #define PASS_MARK 50
  - #define MAX 200
  - #define PI 3.14159
- Symbolic names are sometimes called constant identifiers.
- Since the symbolic names are constants (not variables), they do not appear in declarations.

Statement	Validity	Remark
#define X = 2.5	Invalid	'=' sign is not allowed
# define MAX 10	Invalid	No white space between # and define
#define N 25;	Invalid	No semicolon at the end
#define N 5, M 10	Invalid	A statement can define only one name.
#Define ARRAY 11	Invalid	define should be in lowercase letters
#define PRICE$ 100	Invalid	$ symbol is not permitted in name

DEFINING SYMBOLIC CONSTANTS

- The following rules apply to a #define statement which define a symbolic constant:

1. Symbolic names have the same form as variable names. (Symbolic names are written in **CAPITALS** to visually distinguish them from the normal variable names, which are written in lowercase letters.
2. No blank space between the pound sign '#' and the word define is permitted.
3. '#' must be the first character in the line.
4. A blank space is required between #define and symbolic name and between the symbolic name and the constant.
5. #define statements must not end with a semicolon.
6. After definition, the symbolic name should not be assigned any other value within the program by using an assignment statement. For example, STRENGTH = 200; is illegal.
7. Symbolic names are NOT declared for data types. Its data type depends on the type of constant.
8. #define statements may appear anywhere in the program but before it is referenced in the program (the usual practice is to place them in the beginning of the program).