Basics

Why C

- used to design the system software like
  Operating system
  Compiler
  Network Devices
- To develop application software like database and spread sheets.
- For Develop Graphical related application like computer and mobile games.
- To evaluate any kind of mathematical equation use c language.

Features

- Simple
- Portability
- Powerful
- Platform dependent
- Structure oriented
- Case sensitive
- Compiler based
- Modularity
- Middle level language
- Syntax based language
- Use of Pointers

Compiler

Source Code

- Source code is in the form of Text form.
- Source code is Human Readable Code.
- Source code is Generated by Human or Programmer.
- Source code is receive Compiler as a Input.

Compiler is a system software which converts programming language code into binary format in single steps

Object Code

- Object Code is in the form of Binary Numbers.
- Object Code is in Machine Readable formats.
- Object Code is Generated by Compiler.

- Object Code is Generated by Compiler as a Output.

Hello World

Glimpse

#include <> importing Library

main() Function - The Main function which Compilers Runs

{ } Block

; SemiColon - End Of a line /operation / instruction

return value returned to function as output

printf(); Prints the text on Screen

Refer Stdio.h Header file Programs for printf() function

Program :

```
#include <stdio.h>

int main(){
        printf("Hello World");
        return 0;
}
```

OutPut:

Hello World

```
Hello world!

Process returned 0 (0x0)   execution time : 0.024 s
Press any key to continue.
```

main( ) function

main( ) is a function. Every function has a pair of parentheses ( ) associated with it.

printf( ) function

Syntax : printf ("<format string>", <list of variables>) ;

<format string> can contain -> Format specifiers

%f for printing real values

%d for printing integer values

%c for printing character values

- importing

#include

- it includes a built-in headers files and also user define header files
    - Importing Inbuilt Header Files

        #include <stdio.h>

    - importing User defined Header Files

        #include "My_header.h"

  < > states to compiler to search for header file in default header files location
  " " states to compiler to check for header file in current dir where the c program is located

#define

Used to define Constants , that can't b altered in Program
(genarlly define variables constants are written in CAPS)

#define GRAVITY 9.8

#ifdef, #else and #endif

"#ifdef" directive checks whether particular macro is defined or not. If it is defined, "If" clause statements are included in source file. Otherwise, "else" clause statements are included in source file for compilation and execution.

#if, #else and #endif

28 November 2020 20:54

#if, #else and #endif

"If" clause statement is included in source file if given condition is true. Otherwise, else clause statement is included in source file for compilation and execution.

undef and pragma

28 November 2020 20:56

undef

This directive undefines existing macro in the program. In below program we first undefine age variable and again it define with new value.

pragma

Pragma is used to call a function before and after main function in a C program.

Escape Char " \ "

Escape sequence	Character represented
\a	Alert
\b	Backspace
\e	Escape Char
\f	Form feed Page Break
\n	New Line
\r	Carriage Return
\t	Tab - Horizontal
\v	Tab - Vertical
\\	BackSlash
\'	Single quotation mark
\"	Double quotation mark
\?	Question mark
nnn	The byte whose numerical value is given by *nnn* interpreted as an Octal number
\x*hh…*	The byte whose numerical value is given by *hh…* interpreted as a Hexadecimal number
\u*hhhh*	Unicode code point below 10000 hexadecimal
\U*hhhhhhhh*	Unicode code point where *h* is a hexadecimal digit

Comments

Comments :

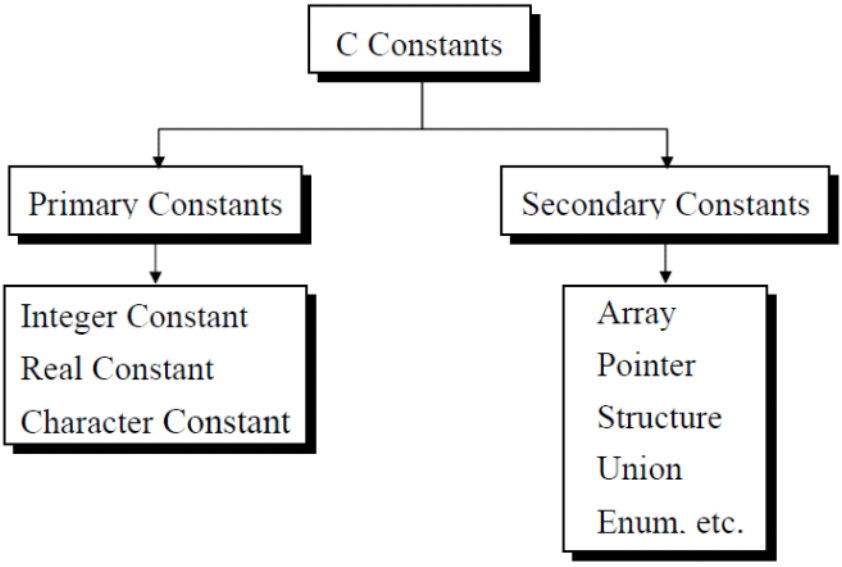
single line comment //

multi line comments /* …….. */

(Comments cannot be nested.)

Data Types

Constants



Data type is a keyword used to identify type of data. Data types are used for storing the input of the Program into the main memory (RAM) of the computer by allocating sufficient amount of memory space in the main memory of the computer.

Data Types in C

- **char**
  The most basic data type in C. It stores a single character and requires a single byte of memory in almost all compilers.

- **int**
  As the name suggests, an int variable is used to store an integer.

- **float**
  It is used to store decimal numbers (numbers with floating point value) with single precision.

- **double**
  It is used to store decimal numbers (numbers with floating point value) with double precision.

DATA TYPE	MEMORY (BYTES)	RANGE	FORMAT SPECIFIER
short int	2	-32,768 to 32,767	%hd
unsigned short int	2	0 to 65,535	%hu
unsigned int	4	0 to 4,294,967,295	%u
int	4	-2,147,483,648 to 2,147,483,647	%d
long int	4	-2,147,483,648 to 2,147,483,647	%ld
unsigned long int	4	0 to 4,294,967,295	%lu
long long int	8	-(2^63) to (2^63)-1	%lld
unsigned long long int	8	0 to 18,446,744,073,709,551,615	%llu
signed char	1	-128 to 127	%c
unsigned char	1	0 to 255	%c
float	4		%f
double	8		%lf
long double	16		%Lf

Variables

An entity that may vary during program execution is called a variable.

Rules :
1.A variable name is any combination of 1 to 31 alphabets, digits or underscores. (Some compilers allow variable names whose length could be up to 247 characters.)
2.The first character in the variable name must be an alphabet or underscore.
3.No commas or blanks are allowed
4.No special symbol other than an underscore.

Declaring

- C compiler is able to distinguish between the variable names by making it compulsory for you to declare the type of any variable name.
- This type declaration is done at the beginning of the program.

Key Words in C

Keyword is a predefined or reserved word in C library with a fixed meaning and used to perform an internal operation. C Language supports 32 keywords.

Keywords are the words whose meaning has already been explained to the C compiler. The keywords are also called 'Reserved words'.

32 Keywords in C Language

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Note that compiler vendors (like Microsoft, Borland, etc.) provide their own keywords apart from the ones mentioned above. These include extended keywords like **near, far, asm**, etc.

Though it has been suggested by the ANSI committee that every such compiler specific keyword should be **preceded by two underscores** (as in __asm), not every vendor follows this rule

Errors in C

Error is a abnormal condition whenever it occurs execution of the program is stopped these are mainly classified into following types.

Types of Error in C

- Compile Time Error
- Run time error

Compile time error

If any error is generated at the time of compilation is known as compile time error, in general these are raised while break down the rules and regulation of programming language.

Run time error

If any error is generated at run time is known as runtime error, in general these are raised because of writing wrong logics in the program.

Warning

Warning is also an abnormal condition but whenever it occurred execution of program will never be stopped.

Type Casting

Type casting is process to convert a variable from one data type to another data type.

Syntax : (Data_type) Variable_Name ;

Example :

CODE : https://github.com/Mahanth-Maha/My_C_Prog/blob/main/typecast.c

OUTPUT :

```
Enter Current Score : 95

Enter Overs Finished (int Only) :15

Current Runrate is 6.33

Process returned 0 (0x0)   execution time : 7.018 s
Press any key to continue.
```

Cmd Line Args

It is a concept to passing the arguments to the main() function by using command prompt.

In Command line arguments application main() function will takes two arguments that is;

- argc
- argv

argc

argc is an integer type variable and it holds total number of arguments which is passed into main function. It take Number of arguments in the command line including program name.

argv[]

argv[] is a char* type variable, which holds actual arguments which is passed to main function in an array.

Syntax : the main() function

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

}
```

Accessing :

in the main() : Can access like a normal variable . argc and argv

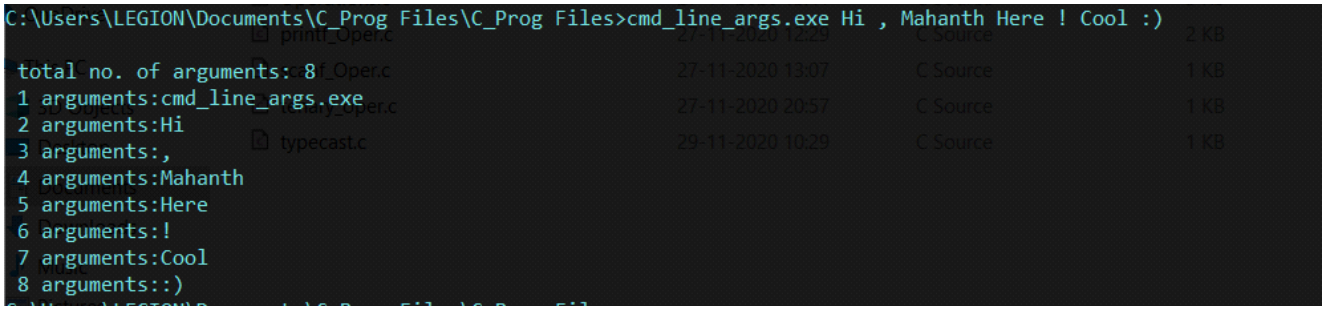
out side the main() : by using _argc, _argv variables.

(since _argc and _argv are global variable which is declared in dos.h .so include dos.h)

Example :

CODE : https://github.com/Mahanth-Maha/My_C_Prog/blob/main/cmd_line_args.c

OUTPUT :



C Instructions

There are basically three types of instructions in C:

- Type Declaration Instruction
- Arithmetic Instruction
- Control Instruction

1. Type Declaration Instruction
   To declare the type of variables used in a C program.
2. Arithmetic Instruction
   To perform arithmetic operations between con-stants and variables
3. Control Instruction
   To control the sequence of execution of various state-ments in a C program.

Type Declaration Instruction

The type declaration statement is written at the beginning of main( ) function.

syntax : <data type> <variable_name> = <value>;

Ex :

```
int My_int = 544;
float My_Float = 66.33;
char My_Char = 'M';
char[5] = "maha"
```

Rules :

[data types] [variable name] ;

Also Supported decalrations

1. Multi var declare - in single line
   Ex : int a=0,b=0,c=0,d=0;

2. Multi Var assigning in single line
   Ex : int a=b=c=d=0;

3. Operations in same line (if and only if var declared before we use it)
   Ex : int a=15 ,b=a+45;
   (but not int b=a + 45 ,b=15;)

Arithmetic Instruction

A C arithmetic instruction consists of a variable name on the left hand side of = and variable names & constants on the right hand side of =.
The variables and constants appearing on the right hand side of = are connected by arithmetic operators

The variables and constants together are called 'operands' that are operated upon by the 'arithmetic operators' and the result is assigned, using the assignment operator, to the variable on left-hand side.

<operand> <assignment operator> <operand> <arthematic operators> <operand> ;

Ex : Total = First_No + Second_No ;

Arithmetic Operators

Operator	Meaning of Operator
+	addition or unary plus
-	subtraction or unary minus
*	multiplication
/	division
%	remainder after division (modulo division)

Increment and Decrement Operators

++	Increment operator increases the integer value by one.
--	Decrement operator decreases the integer value by one.

Assignment Operators

Operator	Same as
=	a = b
+=	a = a+b
-=	a = a-b
*=	a = a*b
/=	a = a/b
%=	a = a%b

Relational Operators

Operator	Meaning of Operator
==	Equal to
>	Greater than
<	Less than
!=	Not equal to
>=	Greater than or equal to
<=	Less than or equal to

Logical Operators

Operator	Meaning
&&	Logical AND. True only if all operands are true
||	Logical OR. True only if either one operand is true
!	Logical NOT. True only if the operand is 0

Bitwise Operators

Operators	Meaning of operators
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise complement
<<	Shift left
>>	Shift right

OUTPUT

```
Operators in C

a = 123 and b = 456

Arithmetic Operators
+ operator : a+b = 579
- operator : b-a = 333
* operator : a*b = 56088
/ operator : b/a = 3
```

Mod operator : b Mod a = 87

Increment and Decrement Operators

a = 123 and b = 456
++ operator : a++ = 123
++ operator : ++b = 457

a = 124 and b = 457
-- operator : a-- = 124
-- operator : --b = 456

a = 123 and b = 456

Assignment Operators

a = 123 and b = 456
= operator : (a =333) = 333

a = 333 and b = 456
+= operator : (b+=44) = 500

a = 333 and b = 500
-= operator : (b-=a) = 167

a = 333 and b = 167
= operator : (a=2) = 666

a = 666 and b = 167
/= operator : (a/=2) = 333

a = 333 and b = 167
Mod= operator : (bMod=21) = 20

a = 333 and b = 20

Relational Operators
True = 1 and False = 0 always

c = 10 , d = 20
== operator : (c==d/2) = 1
!= operator : (c!=d) = 1
> operator : (c>d) = 0
< operator : (c<d) = 1
>= operator : (2*c>=d) = 1
<= operator : (c<=2*d) = 1

Logical Operators
c = 10 , d = 20 , e = 30 , f = 40

&& operator : (2*c==d && 2*c==e) = 0
|| operator : (2*c==d || 2*c==f) = 1

```
! operator : (!(c==f)) = 1

Bitwise Operators
a=20 == 00101 and b=40 == 000101
& operator : a&1 = 0
 | operator : b|1 = 41
^ operator : a^1 = 21
~ operator : b~ = -41
>> operator :a>>1 = 10
<< operator :b<<1 = 80

Process returned 0 (0x0)   execution time : 0.034 s
Press any key to continue.
```

CODE:

https://github.com/Mahanth-Maha/My_C_Prog/blob/main/Operations.c

Control Instruction

As the name suggests the ‘Control Instructions’ enable us to specify the order in which the various instructions in a program are to be executed by the computer.
The control instructions determine the ‘flow of control’ in a program.

There are four types of control instructions in C. They are:

- Sequence Control Instruction
- Selection or Decision Control Instruction
- Repetition or Loop Control Instruction
- Case Control Instruction

Expression Evaluation

Associativity

It represents which operator should be evaluated first if an expression is containing more than one operator with same priority.

Operator	Priority	Associativity
{}, (), []	1	Left to right
++, --, !	2	Right to left
*, /, %	3	Left to right
+, -	4	Left to right
<, <=, >, >=, ==, !=	5	Left to right
&&	6	Left to right
||	7	Left to right
?:	8	Right to left
=, +=, -=, *=, /=, %=	9	Right to left

Storage Classes

Storage class specifiers in C language tells to the compiler where to store a variable (Storage area of variable), how to store the variable, Scope of variable, Default value of a variable (if it is not initialized it), what is the initial value of the variable and life time of the variable. Storage classes of C will provides following information to compiler.

- Storage area of variable
- Scope of variable that is in which block the variable is visible.
- Life time of a variable that is how long the variable will be there in active mode.
- Default value of a variable if it is not initialized it.

Type of Storage Class

Storage classes in mainly divided into four types,

- auto
- extern
- static
- register

Properties of All storage class

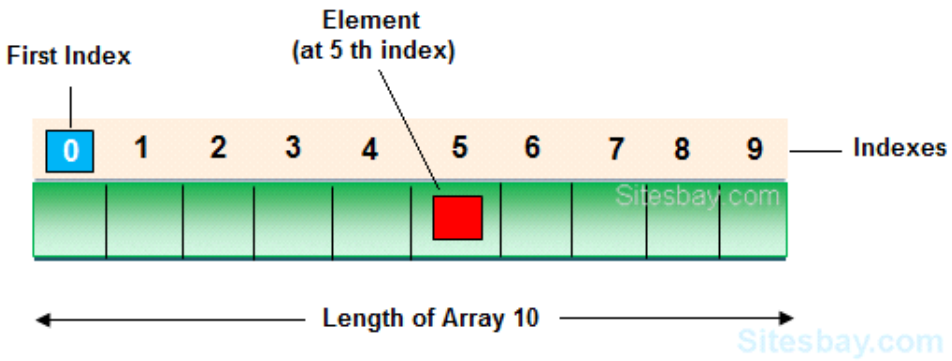
Type	Storage place	Scope	Life	Default Value
auto	CPU Memory	body	Within the Function	Garbage value
static	CPU Memory	function	program	0 (zero)
extern	CPU Memory	program	Till the end of the main program.	0 (zero)
register	Register memory	body	Within the Function	Garbage value

Arrays

Arrays a kind of data structure that can store a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Array Starts from 0

An **Array** is a collection of similar data type value in a single variable. It is a derived data type in C, which is constructed from fundamental data type of C language.



Syntax :
declaring -

data_type Var_Name[Number_of_elements] = { Value1 , Value2, …. ,ValueN };

or data_type Var_Name[Number_of_Ele] ;

Accessing -

Var_Name[Index_of_value];

*index number starts from 0

Example :

CODE : https://github.com/Mahanth-Maha/My_C_Prog/blob/main/array.c

OUTPUT :

```
Enter the Score for ball 1 : 1
Enter the Score for ball 2 : 2
Enter the Score for ball 3 : 3
Enter the Score for ball 4 : 3
Enter the Score for ball 5 : 4
Enter the Score for ball 6 : 0

Score Card : 1 2 3 3 4 0
The total runs in the over is 13 and Strike rate is 216.67
Process returned 0 (0x0)   execution time : 8.004 s
Press any key to continue.
```

2 Dimensional Array

Example :

CODE : https://github.com/Mahanth-Maha/My_C_Prog/blob/main/2darray.c

OUTPUT :

```
Enter the Score for over 3 - ball 3 : 2
Enter the Score for over 3 - ball 4 : 1
Enter the Score for over 3 - ball 5 : 0
Enter the Score for over 3 - ball 6 : 3
Enter the Score for over 4 - ball 1 : 0
Enter the Score for over 4 - ball 2 : 0
Enter the Score for over 4 - ball 3 : 0
Enter the Score for over 4 - ball 4 : 1
Enter the Score for over 4 - ball 5 : 0
Enter the Score for over 4 - ball 6 : 6

Score Card :
Over - 0 : 1 0 2 0 0 1
Over - 1 : 4 1 0 0 6 0
Over - 2 : 4 1 0 0 2 0
Over - 3 : 4 2 2 1 0 3
Over - 4 : 0 0 0 1 0 6


The Total Score is 41
Net Run rate is 6.83

Process returned 0 (0x0)   execution time : 37.414 s
Press any key to continue.
```

3 Dimensional Array

Example :

CODE : https://github.com/Mahanth-Maha/My_C_Prog/blob/main/3darray.c

OUTPUT :

```
Enter the Score for innings 2 over 1 - ball 4 : 0
Enter the Score for innings 2 over 1 - ball 5 : 0
Enter the Score for innings 2 over 1 - ball 6 : 2

Score Card :

Innings 1 Team 1
Over - 1 : 1 2 0 1 3 3
Over - 2 : 4 0 4 4 4 0

The Total Score is 26
Net Run rate is 4.33

Innings 2 Team 2
Over - 1 : 4 4 1 2 4 4
Over - 2 : 3 3 6 0 0 2

The Total Score is 33
Net Run rate is 5.50

winner is team 2

Process returned 0 (0x0)   execution time : 27.274 s
Press any key to continue.
```

Strings in C

String is a collection of character or group of character, it is achieve in C language by using array character. The string in C language is one-dimensional array of character which is terminated by a null character '\0'.

String normally contains a \0 char at the end of the string which is not intialized by us but intialeized by compiler , so we need { <Length Of String> + 1 } memory space to store .

In arrays we give a space of { <length> + 1 } for an String at intializing .

String Variable is a group of chars , so the string var not itself stores the data ,infact str varibale is an pointer to the first char variable .

(So we no need to Pass the & (Address operator) in scanf function to read and store the input.)

if the string length is below the size of array then the remaining values are stored as a NULL value.

Syntax :

Intializing : char Str[<length>+1] = { <length> Chars } ;

Accesing : Same as array

getting the string input : scanf() , gets()

printing the string in console : printf() , puts()

FOR STRING OPERATINS REFER STRING.H

Structure

Structure is a user defined data type which hold or store heterogeneous data item or element in a single variable.
It is a Combination of primitive and derived data type.

Struct

In C language array is also a user defined data type but array hold or store only similar type of data, If we want to store different-different type of data in then we need to defined separate variable for each type of data.

Empty Structure is not possible in C Language.

Can Store Max upto defined . Allocates memory for each and every struct .

```
Syntax : struct Struct_Name {
                data_type Var_Name ;
                .
                .
                .
            } ;
```

```
intitalizing custom struct s
                struct Struct_Name {
                        data_type Var_Name1 ;
                        .
                        .
                        .
                } Custom_Struct_Var_Name1 , Custom_Struct_Var_Name2 , . . .  ;

                OR
```

for Normal variable

intializing : struct Struct_Name Custom_Struct_Var_Name ;

Accessuing : (.) dot operator

Custom_Struct_Var_Name.Var_Name

for Pointer variable

intializing : struct Struct_Name * Custom_Struct_Var_Name ;

Accessuing : (->) Arrow operator

Custom_Struct_Var_Name->Var_Name

Example :

CODE : https://github.com/Mahanth-Maha/My_C_Prog/blob/main/struct.c

OUTPUT :

```
Credentials Storing
Enter Person id : 101
Enter email address of 101 : pranay@pran.in
Enter name of 101 : Pranay
Enter company of 101 : PranaySol

Another Entry (Yes-1/No-0) : 1
Enter Person id : 102
Enter email address of 102 : Vyshnavi776@vyshsol.ltd
Enter name of 102 : Vyshnavi
Enter company of 102 : VyshSol

Another Entry (Yes-1/No-0) : 0
Employee details :

ID :101 Email :pranay@pran.in
Name :Pranay    Company :PranaySol

ID :102 Email :Vyshnavi776@vyshsol.ltd
Name :Vyshnavi  Company :VyshSol


Process returned 0 (0x0)   execution time : 79.903 s
Press any key to continue.
```

Union

A **union** is quite similar to the structures in C. It also store different data types in the same memory location. It is also a user defined data type same like structure.

Syntax : Similar to Structure

The Diiference :

It can store data in one member only.

It occupies less memory because it occupies the memory of largest member only.

```
Syntax : union union_Name {
            data_type Var_Name ;
            .
            .
            .
         } ;
```

intitalizing custom union s

```
            struct union_Name {
                data_type Var_Name1 ;
                .
                .
                .
            } Custom_union_Var_Name1 , Custom_union_Var_Name2 , . . . ;
```

OR

for Normal variable

intializing : union union_Name Custom_union_Var_Name ;

Accessuing : (.) dot operator

Custom_union_Var_Name.Var_Name

for Pointer variable

intializing : union union_Name * Custom_union_Var_Name ;

Accessuing : (->) Arrow operator

Custom_union_Var_Name->Var_Name

Difference between Structure and Union

Difference between Structure and Union

	Structure	Union
1	For defining structure use struct keyword.	For defining union we use union keyword
2	Structure occupies more memory space than union.	Union occupies less memory space than Structure.
3	In Structure we can access all members of structure at a time.	In union we can access only one member of union at a time.
4	Structure allocates separate storage space for its every members.	Union allocates one common storage space for its all members. Union find which member need more memory than other member, then it allocate that much space

enum

An **enum** is a keyword, it is an user defined data type. All properties of integer are applied on Enumeration data type so size of the enumerator data type is 2 byte. It work like the Integer.
It is used for creating an user defined data type of integer. Using enum we can create sequence of integer constant value.

Syntax : enum Var_name {value1, value2, value3,....} ;

Example :

CODE :

OUTPUT :

Decision Control

C has three major decision making instructions—the if statement, the if-else statement, and the switch statement.
A fourth, somewhat less important structure is the one that uses conditional operators.

As a general rule, we express a condition using C's 'relational' operators.
The relational operators allow us to compare two values to see whether they are equal to each other, unequal, or whether one is greater than the other.

Hierarchy of Operators

The higher the position of an operator is in the table, higher is its priority.

Operators	**Type**
!	Logical NOT
* / %	Arithmetic and modulus
+ -	Arithmetic
< > <= >=	Relational
== !=	Relational
&&	Logical AND
||	Logical OR
=	Assignment

if statement

27 November 2020 15:47

Generally

```
if ( this condition is true )
        execute this statement ;
```

Syntax :

```
if ( condition )
        statement ;
```

Nested If

```
if ( condition )
        if ( condition )
                statement ;

        if ( condition )
                statement ;

        .
        .
        .
        .
.
```

Example

CODE : https://github.com/Mahanth-Maha/My_C_Prog/blob/main/if_stmt.c

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
   int smoke;
   printf("do you smoke (Y/N) : ");
   scanf("%c",&smoke);

   // single if statements
   if(smoke =='y' || smoke == 'Y')
      printf("Stop smoking ASAP ,smoking kills\n\n");

   //nested if statements
```

```
    if (smoke == 'y' || smoke == 'Y'){
        if(smoke == 'y'){
            printf("I asked to enter Y or N , but u entered y , why ? :\/\n");
        }
    }

    return 0;
}
```

OUTPUT :

```
do you smoke (Y/N) : y
Stop smoking ASAP ,smoking kills

I asked to enter Y or N , but u entered y , why ? :/

Process returned 0 (0x0)   execution time : 3.763 s
Press any key to continue.
```

if - else

27 November 2020 15:47

The group of statements after the if upto and not including the else is called an 'if block'.
Similarly, the statements after the else form the 'else block'.
Notice that the else is written exactly below the if.
Had there been only one statement to be executed in the if block and only one statement in the else block we could have dropped the pair of braces

Syntax : if (condition)

```
{
    statement;
}
else
{
    statement;
}
```

Nested if-elses

```
if ( condition )
        do this ;
else{
        if ( condition )
                do this ;
        else
        {
                do this ;
        }
}
```

Example : https://github.com/Mahanth-Maha/My_C_Prog/blob/main/if_else_stmt.c

CODE:

```
#include <stdio.h>
#include <stdlib.h>

void binP(int);
int main()
{
   char license='n',llr='n',supervision='n';
   printf("do you Have a Driving license (Y/N) : ");
   scanf(" %c",&license);

   // single if statements
   if(license =='y' || license == 'Y')
      printf("\nYou Can Drive !\n");
```

```
    else{
        printf("\nDo You Have LLR (Y/N) : ");
        scanf(" %c",&llr);
    }
    //nested if statements
    if(llr == 'y' || llr == 'Y'){
        printf("\nDo You Have any SuperVision (Y/N) : ");
        scanf(" %c",&supervision);
        if(supervision == 'y' || supervision == 'Y'){
            printf("You Can Drive ");
          }
        else{
          printf("You Can Drive only when there is a supervision!\n");
        }
    }else{
        if(!(license =='y' ||license == 'Y')){
          printf("\nYou Can't Drive ,Until You get an LLR !\n");
        }
    }


    return 0;
}
```

OUTPUT:

```
do you Have a Driving license (Y/N) : n

Do You Have LLR (Y/N) : y

Do You Have any SuperVision (Y/N) : n
You Can Drive only when there is a supervision!

Process returned 0 (0x0)   execution time : 8.861 s
Press any key to continue.
```

else - if

27 November 2020 15:47

extended form of if -else

can have many if statements or can check many conditions before entering into else

Syntax :
```
if (condition) {
        statements;
}
else if (condition){
        statements;
}
else if (condition){
        statements;
}
        .
        .
        .
        .
else{
        statements;
}
```

Example : https://github.com/Mahanth-Maha/My_C_Prog/blob/main/else_if_stmt.c

CODE
```
#include <stdio.h>
#include <stdlib.h>

int main()
{
   int age;
   printf("Enter Your age : ");
   scanf(" %d",&age);
   //How to give ads to certain people
   if(age > 25){
      printf("You probably finished your Degree !\nWe will be showing some
home products\n");
   }
   else if(age>18){
      printf("you probably studying your Degree! \nWe Will offer you some
Personality dev. programs\n");
   }
   else if(age>14){
      printf("You are Probably a College Student \nWe will show you some college
stationarys items\n");
   }
   else if(age>8){
      printf("You are Probably a School Student \nWe will Offer some School Bags
```

```
and books\n");
    }
    else{
        printf("You are child\nWe will offer some toys and chocolates\n");
    }
    return 0;
}
```

OUTPUT:

```
Enter Your age : 23
you probably studying your Degree!
We Will offer you some Personality dev. programs

Process returned 0 (0x0)   execution time : 3.381 s
Press any key to continue.
```

```
Enter Your age : 5
You are child
We will offer some toys and chocolates

Process returned 0 (0x0)   execution time : 4.958 s
Press any key to continue.
```

conditional (?:) operator

27 November 2020 15:47

The conditional operators ? and : are sometimes called ternary operators since they take three arguments.

Syntax : (condition) ? (Statement when conditon True) : (Statement when conditon False) ;

simply : (condition) ? (if statements) : (else statements) ;

used when only single line statements are to be executed , not for multi-line statements

Example:

CODE: https://github.com/Mahanth-Maha/My_C_Prog/blob/main/tenary_oper.c

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
   int age;
   printf("Enter Your age : ");
   scanf(" %d",&age);
   //Checking for Right to Vote
   (age<=17)?printf("\nYou Can't vote ! ( under 18 can't vote)"):printf("\nYou Can
vote !");

   return 0;
}
```

OUTPUT:

```
Enter Your age : 17

You Can't vote ! ( under 18 can't vote)
Process returned 0 (0x0)   execution time : 4.389 s
Press any key to continue.
```

```
Enter Your age : 18

You Can vote !
Process returned 0 (0x0)   execution time : 1.965 s
Press any key to continue.
```

Loop Control

This involves repeating some portion of the program either a specified number of times or until a particular condition is being satisfied.

There are three methods by way of which we can repeat a part of a program. They are:

1. Using a for statement
2. Using a while statement
3. Using a do-while statement

For Loop

Syntax :

```
for ( initialize counter ; test counter ; increment counter )
{
        Statements ;
}
```

Nested for loops are also possible.

```
for ( initialize counter ; test counter ; increment counter )
{
        Statements ;
        for ( initialize counter ; test counter ; increment counter )
        {
                Statements ;
        }
        Statements ;

}
```

Multiple Initializations in the for Loop

```
for ( initialise_counter1 , initialise_counter2  ; test counter ; increment counter )
{
        Statements ;
}
```

Example :

CODE : https://github.com/Mahanth-Maha/My_C_Prog/blob/main/for_loop.c

OUTPUT :

```
----Do You Know----
if we fill the chess board by twice with grains for every box then the list will become :


Basically printing power of 2s)

1       1
2       2
3       4
4       8
5       16
6       32
7       64
8       128
9       256
10      512
11      1024
12      2048
13      4096
14      8192
15      16384
16      32768
17      65536
18      131072
```

While Loop

Syntax : while (condition)

```
{
    Statements ;
    iterator;
}
```

Nested for loops are also possible.

```
while (condition)
{
    while (condition)
    {
        Statements ;
        iterator;
    }
    Statements ;
    iterator;
}
```

Example :

CODE : https://github.com/Mahanth-Maha/My_C_Prog/blob/main/while_loop.c

OUTPUT :

```
Hacking Nasa - Init !!!

Hacking Nasa...0 %
Hacking Nasa...20 %
Hacking Nasa...40 %
Hacking Nasa...60 %
Hacking Nasa...80 %
Hacking Nasa...100 %
Nasa Has been Hacked !


If didn't get the access , init again(Y/N) : n

Process returned 0 (0x0)   execution time : 46.845 s
Press any key to continue.
```

do - while loop

Syntax : do

```
{
        statements ;
        Iterator;

} while ( condition ) ;
```

Example :

CODE : https://github.com/Mahanth-Maha/My_C_Prog/blob/main/do_loop.c

OUTPUT :

```
                Timer

Enter time to count down in sec : 5
5
4
3
2
1

TimeUp !

Process returned 0 (0x0)   execution time : 13.962 s
Press any key to continue.
```

break and continue

Break	**Stop the current loop (exit from the current total loop)**
Continue	**Skip current for a that instant (stop current iteration of loop only not full loop)**

Break

- jump out

break is encountered inside any loop, control automatically passes to the first statement after the loop. A break is usually associated with an if.

Continue

- bypassing

When continue is encountered inside any loop, control automatically passes to the beginning of the loop.
A continue is usually associated with an if.

Example :

CODE : https://github.com/Mahanth-Maha/My_C_Prog/blob/main/break_n_continue.c

OUTPUT :

```
Break is to Stop
Continue is to Skip
1       2       3       4       5       6       7       Breaking at 7
1       2       4       5       7       8       10      Skipping 3 multiples

Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```

Case Control

Basically used to replace if-else-if s and make use of less code and more decisions

1. goto
2. switch - case - default

goto

A goto statement in C programming provides an unconditional jump from the 'goto' to a labeled statement in the same function.

The only programming situation in favour of using goto is when we want to take the control out of the loop that is contained in several other loops.

Example :

CODE : https://github.com/Mahanth-Maha/My_C_Prog/blob/main/goto.c

OUTPUT :

```
Do you want to take quiz(Y/N)
n
Actually , Participation is more than wining so you won !
Process returned 0 (0x0)   execution time : 3.905 s
Press any key to continue.
```

```
Do you want to take quiz(Y/N)
y
Quiz...Loading
Actually , Participation is more than wining so you won !
Process returned 0 (0x0)   execution time : 3.155 s
Press any key to continue.
```

Switch

The control statement that allows us to make a decision from the number of choices is called a switch-case-default,

since these three keywords go together to make up the control statement.

also allowed to use char values in case and switch

They most often appear as follows:

Syntax : switch (integer/char expression)

```
{
        case constant 1 :
                do this ;
        case constant 2 :
                do this ;
        case constant 3 :
                do this ;
        default :
                do this ;
}
```

Example :

CODE : https://github.com/Mahanth-Maha/My_C_Prog/blob/main/switch.c

OUTPUT :

```
Hi College Student Enter Your age : 15
Are U Kidding Me - Only for College Students

Process returned 0 (0x0)   execution time : 4.638 s
Press any key to continue.
```

```
Hi College Student Enter Your age : 21
You are Probably 3 rd Year

Process returned 0 (0x0)   execution time : 2.525 s
Press any key to continue.
```

Functions

A function is a self-contained block of statements that perform a coherent task of some kind. Every C program can be thought of as a collection of these functions

— Any C program contains at least one function.
— If a program contains only one function, it must be main( ).
— If a C program contains more than one function, then one (and only one) of these functions must be main( ), because program execution always begins with main( ).
— There is no limit on the number of functions that might be present in a C program.
— Each function in a program is called in the sequence specified by the function calls in main( ).

```
Syntax : return_type function_name ( parameters )
        {
                Statements ;
                return Value ;
        }
```

Function Declarations

Syntax : return_type funtion_name (Parameters list) ;

Calling a Function

Syntax : funtion_name (arguments) ;

Function Arguments

list of arguments

Call Type	Description
Call by Value	This method copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.
Call by Reference	This method copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the argument.

By default, C uses call by value to pass arguments.
In general, it means the code within a function cannot alter the arguments used to call the function.

Example : https://github.com/Mahanth-Maha/My_C_Prog/blob/main/call_func_type.c

Scope Of Variables

There are three places where variables can be declared in C programming language –

- Inside a function or a block which is called local variables.
- Outside of all functions which is called global variables.
- In the definition of function parameters which are called formal parameters.

Local

Variables that are declared inside a function or block are called local variables. They can be used only by statements that are inside that function or block of code.

Global

Global variables are defined outside a function, usually on top of the program. Global variables hold their values throughout the lifetime of your program and they can be accessed inside any of the functions defined for the program.

Formal

Formal parameters, are treated as local variables with-in a function and they take precedence over global variables.

Example : https://github.com/Mahanth-Maha/My_C_Prog/blob/main/var_scope.c

OUTPUT :

```
In main
        defined Var 'a' here as 544 - value = 544
In func_a_declared
        defined Var 'a' here as 44  - value = 44
In func_a_not_decl
        not defined Var 'a' here inherited from main - value = 544

Process returned 0 (0x0)   execution time : 0.557 s
Press any key to continue.
```

Recursion

it is possible for the functions to call themselves. A function is called 'recursive' if a statement within the body of a function calls the same function. Sometimes called 'circular definition', recursion is thus the process of defining something in terms of itself.

Syntax : return_type function_name(parameters)

```
{
        Statements ;
        functoin_name(args);
}
```

Example : https://github.com/Mahanth-Maha/My_C_Prog/blob/main/recursive_func.c

OUTPUT :

```
Enter a integer to find out power of 2 recursively : 8
in rec_func( 8 )
in rec_func( 7 )
in rec_func( 6 )
in rec_func( 5 )
in rec_func( 4 )
in rec_func( 3 )
in rec_func( 2 )
in rec_func( 1 )
in rec_func( 0 )

2 power 8 = 256

Process returned 0 (0x0)   execution time : 1.621 s
Press any key to continue.
```

STDIO.H

LIST OF INBUILT C FUNCTIONS IN STDIO.H FILE:

Function	Description
printf()	This function is used to print the character, string, float, integer, octal and hexadecimal values onto the output screen
scanf()	This function is used to read a character, string, numeric data from keyboard.
getc()	It reads character from file
gets()	It reads line from keyboard
getchar()	It reads character from keyboard
puts()	It writes line to o/p screen
putchar()	It writes a character to screen
clearerr()	This function clears the error indicators
f open()	All file handling functions are defined in stdio.h header file
f close()	closes an opened file
getw()	reads an integer from file
putw()	writes an integer to file
f getc()	reads a character from file
putc()	writes a character to file
f putc()	writes a character to file
f gets()	reads string from a file, one line at a time
f puts()	writes string to a file
f eof()	finds end of file
f getchar	reads a character from keyboard
f getc()	reads a character from file
f printf()	writes formatted data to a file
f scanf()	reads formatted data from a file
f getchar	reads a character from keyboard
f putchar	writes a character from keyboard
f seek()	moves file pointer position to given location
SEEK_SET	moves file pointer position to the beginning of the file
SEEK_CUR	moves file pointer position to given location
SEEK_END	moves file pointer position to the end of file.
f tell()	gives current position of file pointer
rewind()	moves file pointer position to the beginning of the file
putc()	writes a character to file
sprint()	writes formatted output to string

sscanf()	Reads formatted input from a string
remove()	deletes a file
fflush()	flushes a file

printf( )

printf()

This function is used to print the character, string, float, integer, octal and hexadecimal values onto the output screen

Syntax : printf ("<format string>", <list of variables>) ;

- format tags prototype is %[flags][width][.precision][length]specifier

sprint()

Writes formatted output to string

format specifiers

Format Specifier	Type
%c	Character
%d	Signed integer
%e or %E	Scientific notation of floats
%f	Float values
%g or %G	Similar as %e or %E
%hi	Signed integer (short)
%hu	Unsigned Integer (short)
%i	Unsigned integer
%l or %ld or %li	Long
%lf	Double
%Lf	Long double
%lu	Unsigned int or unsigned long
%lli or %lld	Long long
%llu	Unsigned long long
%o	Octal representation
%p	Pointer
%s	String
%u	Unsigned int
%x or %X	Hexadecimal representation
%n	Prints nothing
%%	Prints % character

OUT PUT

```
printf() function
Format Specifiers

M = M : c - Char
544 <-signed-> -44 : d - signed integer
544 : i - Unsigned integer
13.18 = 1.318000e+001           : e - scientific notation
13.18 = 13.180000               : f - float value
44 = 54                 : o - octal format
44 = 2c                 : x - hex format
Mahanth Maha : s - Full String
Mahanth Maha : #s - shift cursor to the right #(9) characters
Mahanth Maha : -#s - shift cursor to the left #(9) characters
    Mahan : #.$s - shift cursor to the right #(9) characters and Print string up to $ chars
Mahan     : -#.$s - shift cursor to the left #(9) characters and Print string up to $ chars


Process returned 0 (0x0)   execution time : 0.508 s
Press any key to continue.
```

CODE

https://github.com/Mahanth-Maha/My_C_Prog/blob/main/printf_Oper.c

scanf( )

scanf( )

This function is used to read a character, string, numeric data from keyboard.
This function is a counter-part of the printf( ) function.

printf( ) outputs the values to the screen whereas scanf( ) receives them from the keyboard.

Syntax : scanf ("<format string>", <list of address of variables>) ;

format strings specifiers are same as [printf format specifiers]

for strings

generally the first letter of the string is pointed by the variable(i.e, the address of first letter is stored in variable name)

- so for strings we just pass variable name to read (No & required)

for other var (other than strings)

we need to pass the address of the variable to store the input value so , we use & (address opertor) to specify the address of variable .

- we use & in front of variables

sscanf()

Reads formatted input from a string

OUTPUT

```
scanf() function
Reading user input

Enter a Char : M
The Read Char Value = M         : c - read single char
Enter a integer value : 544
The Read int Value = 544        : d - read integer
Enter a float value : 123.456789123
The Read float Value = 123.456787       : f - read float
Enter a stirng : Mahanth Maha
The Read int Value = Mahanth    : s - read string(first word up to space or an enter[new line])

Process returned 0 (0x0)   execution time : 28.384 s
Press any key to continue.
```

CODE

https://github.com/Mahanth-Maha/My_C_Prog/blob/main/scanf_Oper.c

get & put

GET

gets()

It reads line from keyboard

getchar()

It reads character from keyboard

PUT

puts()

It writes line to o/p screen

putchar()

It writes a character to screen

OUTPUT

Char

```
read the char with getchar and print it with putchar

Enter a Char : ^
You Entered Char is ^
Process returned 0 (0x0)   execution time : 8.202 s
Press any key to continue.
```

Str

```
read the str with gets and print it with puts

Enter a String : Mahanth Maha Codes like a Dumb
You Entered String is Mahanth Maha Codes like a Dumb

Process returned 0 (0x0)   execution time : 41.618 s
Press any key to continue.
```

CODE

https://github.com/Mahanth-Maha/My_C_Prog/blob/main/get_n_put_char.c
https://github.com/Mahanth-Maha/My_C_Prog/blob/main/get_n_put_str.c

STDLIB.H

Function	Description
malloc()	This function is used to allocate space in memory during the execution of the program.
calloc()	This function is also like malloc () function. But calloc () initializes the allocated memory to zero. But, malloc() doesn't
realloc()	This function modifies the allocated memory size by malloc () and calloc () functions to new size
free()	This function frees the allocated memory by malloc (), calloc (), realloc () functions and returns the memory to the system.
abs()	This function returns the absolute value of an integer . The absolute value of a number is always positive. Only integer values are supported in C.
div()	This function performs division operation
abort()	It terminates the C program
exit()	This function terminates the program and does not return any value
system()	This function is used to execute commands outside the C program.
atoi()	Converts string to int
atol()	Converts string to long
atof()	Converts string to float
strtod()	Converts string to double
strtol()	Converts string to long
getenv()	This function gets the current value of the environment variable
setenv()	This function sets the value for environment variable
putenv()	This function modifies the value for environment variable
perror()	This function displays most recent error that happened during library function call.
rand()	This function returns the random integer numbers
delay()	This function Suspends the execution of the program for particular time

STRING.H

String functions	Description
strcat ( )	Concatenates str2 at the end of str1
strncat ( )	Appends a portion of string to another
strcpy ( )	Copies str2 into str1
strncpy ( )	Copies given number of characters of one string to another
strlen ( )	Gives the length of str1
strcmp ( )	Returns 0 if str1 is same as str2. Returns <0 if strl < str2. Returns >0 if str1 > str2
strcmpi ( )	Same as strcmp() function. But, this function negotiates case. "A" and "a" are treated as same.
strchr ( )	Returns pointer to first occurrence of char in str1
strrchr ( )	last occurrence of given character in a string is found
strstr ( )	Returns pointer to first occurrence of str2 in str1
strrstr ( )	Returns pointer to last occurrence of str2 in str1
strdup ( )	Duplicates the string
strlwr ( )	Converts string to lowercase
strupr ( )	Converts string to uppercase
strrev ( )	Reverses the given string
strset ( )	Sets all character in a string to given character
strnset ( )	It sets the portion of characters in a string to given character
strtok ( )	Tokenizing given string using delimiter
memset()	It is used to initialize a specified number of bytes to null or any other value in the buffer
memcpy()	It is used to copy a specified number of bytes from one memory to another
memmove()	It is used to copy a specified number of bytes from one memory to another or to overlap on same memory.
memcmp()	It is used to compare specified number of characters from two buffers
memicmp()	It is used to compare specified number of characters from two buffers regardless of the case of the characters
memchr()	It is used to locate the first occurrence of the character in the specified string

TIME.H

Functions	Description
setdate()	This function used to modify the system date
getdate()	This function is used to get the CPU time
clock()	This function is used to get current system time
time()	This function is used to get current system time as structure
difftime()	This function is used to get the difference between two given times
strftime()	This function is used to modify the actual time format
mktime()	This function interprets tm structure as calendar time
localtime()	This function shares the tm structure that contains date and time information
gmtime()	This function shares the tm structure that contains date and time information
ctime()	This function is used to return string that contains date and time information
asctime()	Tm structure contents are interpreted by this function as calendar time. This time is converted into string.

MATH.H

Function	Description
floor ( )	This function returns the nearest integer which is less than or equal to the argument passed to this function.
round ( )	This function returns the nearest integer value of the float/double/long double argument passed to this function. If decimal value is from ".1 to .5", it returns integer value less than the argument. If decimal value is from ".6 to .9", it returns the integer value greater than the argument.
ceil ( )	This function returns nearest integer value which is greater than or equal to the argument passed to this function.
sin ( )	This function is used to calculate sine value.
cos ( )	This function is used to calculate cosine.
cosh ( )	This function is used to calculate hyperbolic cosine.
exp ( )	This function is used to calculate the exponential "e" to the xth power.
tan ( )	This function is used to calculate tangent.
tanh ( )	This function is used to calculate hyperbolic tangent.
sinh ( )	This function is used to calculate hyperbolic sine.
log ( )	This function is used to calculates natural logarithm.
log10 ( )	This function is used to calculates base 10 logarithm.
sqrt ( )	This function is used to find square root of the argument passed to this function.
pow ( )	This is used to find the power of the given number.
trunc.(.)	This function truncates the decimal value from floating point value and returns integer value.

Pointer

A **pointer** is a variable which contains or hold the address of another variable.
We can create pointer variable of any type of variable.

Symbol	Name	Description
& (ampersand sign)	Address of operator	Give the address of a variable
* (asterisk sign)	Indirection operator	Gives the contents of an object pointed to by a pointer.

Var -----> Value of variable
&Var -----> Address of Varibale
Var_Ptr -----> Address of Varibale
&Var_Ptr ----> Address of Pointer Varibale

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int Var=18;
    int *ptr;
    ptr = &Var;
    printf(" %d \n",Var);
    printf(" %d \n",&Var);
    printf(" %d \n",ptr);
    printf(" %d \n",&ptr);

    return 0;
}
```

```
18
6422044
6422044
6422032

Process returned 0 (0x0)   execution time : 2.987 s
Press any key to continue.
```

Advantage of pointer

- Pointer reduces the code and improves the performance, because it direct access the address of variable.
- Using pointer concept we can return multiple value from any function.
- Using pointer we can access any memory location from pointer.

Syntax :

Declaring : data_type * (pointer_Var_Name) ;

Intializing :

Assiging : Pointer_Var_Name = & (variable_name) ;

Assign to Nothing : Pointer_Var_Name = NULL ;

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int a=544,*pa;
    printf("created integers : a = %d , pa \n\n",a);
    pa = &a;
    printf("Linking address of a to pa as pa = &a : a = %d ,pa =%d",a,pa);
    printf("\nValue of a\t - a = %d",a);
    printf("\naddress of a\t - &a = %d",&a);
    printf("\nValue of pa\t - pa = %d",pa);
    printf("\naddress of pa\t - &a = %d",&pa);

    return 0;
}
```

```
created integers : a = 544 , pa

Linking address of a to pa as pa = &a : a = 544 ,pa =6422044
Value of a        - a = 544
address of a      - &a = 6422044
Value of pa       - pa = 6422044
address of pa     - &a = 6422032
Process returned 0 (0x0)   execution time : 4.206 s
Press any key to continue.
```

Type Def

The C programming language provides a keyword called **typedef**, by using this keyword you can create a user defined name for existing data type. Generally typedef are use to create an **alias name** (nickname).

Syntax : typedef datatype alias_name;

Example :

CODE :

OUTPUT :

```
declared a as int with int a = 544
defined int as i and declaring b as int with i
as i b = 999

adding both  and to ensure same type : 544 + 999 = 1543

Process returned 0 (0x0)   execution time : 1.607 s
Press any key to continue.
```

Buffer in C

Temporary storage area is called buffer.
All standard input output devices are containing input output buffer.

In implementation when we are passing more than required number of values as a input then rest of all values will automatically holds in standard input buffer, this buffer data will automatically pass to next input functionality if it is exist.

In implementation when we need to remove standard input buffer data then go for flushall() or fflush() function.

flushall()

it is a predefined function which is declared in stdio.h.
by using flushall we can remove the data from standard input output buffer.

fflush()

it is a predefined function which is declared in stdio.h.
used to flush or clear either input or output buffer memory.

fflush(stdin)

it is used to clear the input buffer memory. It is recommended to use before writing scanf statement.

fflush(stdout)

it is used to clear the output buffer memory. It is recommended to use before printf statement.

Dynamic Memory Allocation

- It is a process of allocating or de-allocating the memory at run time it is called as dynamically memory allocation.

  Dynamic memory allocation related all predefined functions are declared in following header files.

  - <alloc.h>
  - <malloc.h>
  - <mem.h>
  - <stdlib.h>

- By using malloc(), calloc(), realloc() we can create maximum of 64kb data only.
- In implementation when we need to create more than 64kb data then go for formalloc(), forcalloc() and forrealloc().
- By using free() we can de-allocate 64kb data only,
- if we need to de-allocate more than 64kb data then

  forfree().
  formalloc()
  voidfor*formalloc(size-type);

malloc( )

By using malloc() we can create the memory dynamically at initial stage.
Malloc() required one argument of type size type that is data type size malloc() will creates the memory in bytes format.
Malloc() through created memory initial value is garbage.

Dynamic memory allocation related function can be applied for any data type ,
intially returns : void*
type casting is required.

Syntax : (data_type_to_get) malloc (No_of_vars_req * sizeof(data_type)) ;

calloc( )

By using calloc() we can create the memory dynamically at initial stage.
calloc() required 2 arguments of type count, size-type.

Count will provide number of elements;

size-type is data type size.

calloc() will creates the memory in blocks format.

Initial value of the memory is zero.

Syntax : (data_type_to_get) calloc (No_of_vars_req , sizeof(data_type)) ;

realloc( ) & free( )

realloc ( )

- By using realloc() we can create the memory dynamically at middle stage.
- Generally by using realloc() we can reallocation the memory.
- Realloc() required 2 arguments of type void*, size_type.
- Void* will indicates previous block base address, size-type is data type size.
- Realloc() will creates the memory in bytes format and initial value is garbage.

Syntax : (data_type_to_get) malloc (var_name , sizeof(Var_Name)) ;

OR (data_type_to_get) malloc
(Var_Name ,sizeof(data_type)*No_of_vars_in_var) ;

free ( )

- dynamic memory allocation related memory is a permanent memory if we are not de-allocated
  - so it is always recommended to deleted the memory at the end of the program.
- By using free( ) dynamic allocation memory can be de-allocated.
- free() requires one arguments of type void*

Syntax : free (Var_Name_allocated_dyn) ;

File Handling

File Handling concept in C language is used for store a data permanently in computer.
Using this concept we can store our data in Secondary memory (Hard disk).
All files related function are available in **stdio.h** header file.

Function for file operations

Function	Description
fopen()	opens new or existing file
fprintf()	write data into the file
fscanf()	reads data from the file
fputc()	writes a character into the file
fgetc()	reads a character from file
fclose()	closes the file
fseek()	sets the file pointer to given position
fputw()	writes an integer to file
fgetw()	reads an integer from file
ftell()	returns current position
rewind()	sets the file pointer to the beginning of the file

File pointer

SEEK_SET	moves file pointer position to the beginning of the file
SEEK_CUR	moves file pointer position to given location
SEEK_END	moves file pointer position to the end of file.

Modes in file

Mode	Description
r	opens a text file in read mode
w	opens a text file in write mode
a	opens a text file in append mode
r+	opens a text file in read and write mode
w+	opens a text file in read and write mode
a+	opens a text file in read and write mode

rb	opens a binary file in read mode
wb	opens a binary file in write mode
ab	opens a binary file in append mode
rb+	opens a binary file in read and write mode
wb+	opens a binary file in read and write mode
ab+	opens a binary file in read and write mode

Example : https://github.com/Mahanth-Maha/My_C_Prog/blob/main/fileOpr_csv.c

OUTPUT :

```
Creating a Data Base in CSV
created or opened the file

CAUTION : If you are running this for first Time
        Then Enter Headings rather than Data

Append a Student to CSV (1 = append/0 = No - Exit)? : 1

Enter the Student RollNo : ROLL NO

Enter the Student Name : NAME

Enter the Student Branch : BRANCH

Append a Student to CSV (1 = append/0 = No - Exit)? : 1

Enter the Student RollNo : 101

Enter the Student Name : Tyler

Enter the Student Branch : Risk Management

Append a Student to CSV (1 = append/0 = No - Exit)? : 1

Enter the Student RollNo : 102

Enter the Student Name : Matt

Enter the Student Branch : Data Management

Append a Student to CSV (1 = append/0 = No - Exit)? : 0

closing csv file

Process returned 0 (0x0)   execution time : 116.426 s
Press any key to continue.
```

RESULT :

	A	B	C	D	E
1	ROLL NO	NAME	BRANCH		
2	101	Tyler	Risk Management		
3	102	Matt	Data Management		
4					
5					
6					
7					

Storing a table with CSV and File operations from C Programming

CODE :https://github.com/Mahanth-Maha/My_C_Prog/blob/main/csv_Table_Opr.c