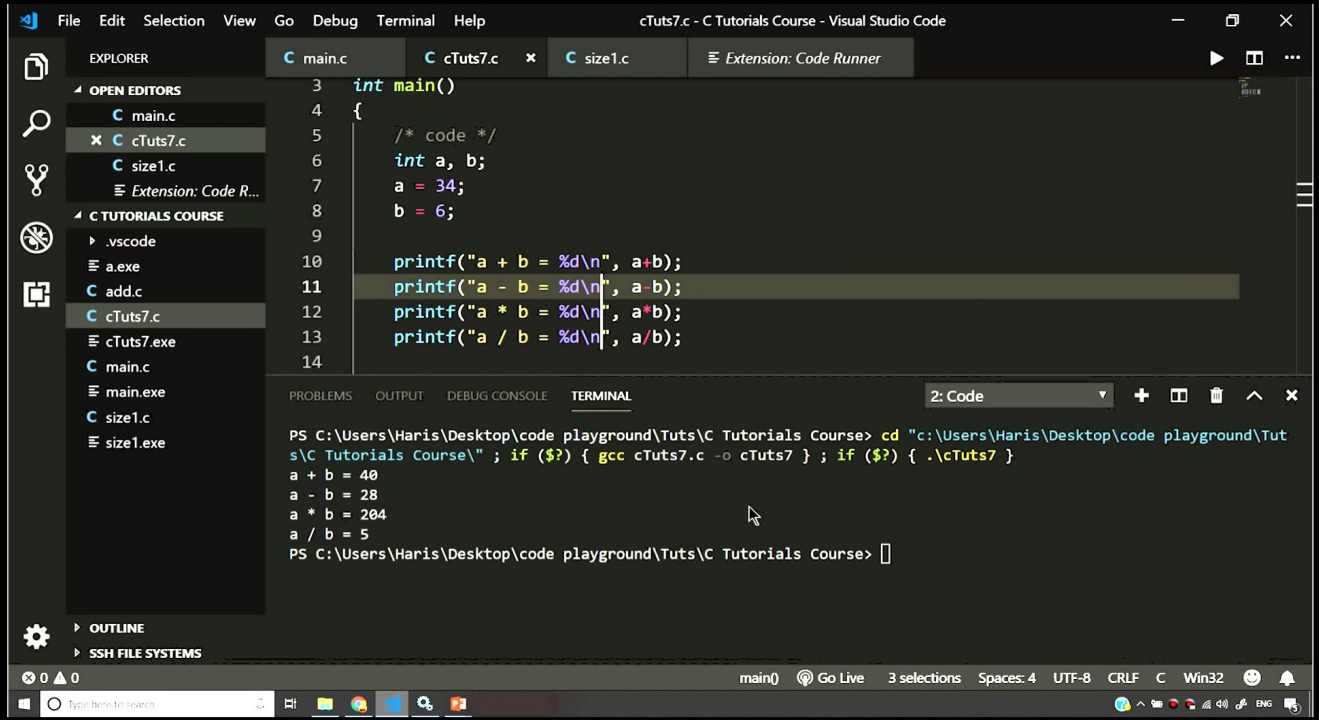
DATA TYPES IN C

- Basic Data Type: int, char, float, double
- Derived Data Type: array, pointer, structure, union
- Enumeration Data Type: emum
- Void Data Type: void

THE C PROGRAMMING LANGUAGE


File Edit Selection View Go Debug Terminal Help
cTuts7.c - C Tutorials Course - Visual Studio Code
EXPLORER
main.c
cTuts7.c
size1.c
Extension: Code Runner
OPEN EDITORS
main.c
cTuts7.c
size1.c
Extension: Code R...
C TUTORIALS COURSE
.vscode
a.exe
add.c
cTuts7.c
cTuts7.exe
main.c
main.exe
size1.c
size1.exe
3 int main()
4 {
5 /* code */
6 int a, b;
7 a = 34;
8 b = 6;
9
10 printf("a + b = %d\n", a+b);
11 printf("a - b = %d\n", a-b);
12 printf("a * b = %d\n", a*b);
13 printf("a / b = %d\n", a/b);
14
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
2: Code
PS C:\Users\Haris\Desktop\code playground\Tuts\C Tutorials Course> cd "c:\Users\Haris\Desktop\code playground\Tuts\C Tutorials Course\" ; if ($?) { gcc cTuts7.c -o cTuts7 } ; if ($?) { .\cTuts7 }
a + b = 40
a - b = 28
a * b = 204
a / b = 5
PS C:\Users\Haris\Desktop\code playground\Tuts\C Tutorials Course>
OUTLINE
SSH FILE SYSTEMS
0 0
main() Go Live 3 selections Spaces: 4 UTF-8 CRLF C Win32
Type here to search
ENG
Subscribe

RELATIONAL OPERATORS

Operator	Description
==	Is equal to
!=	Is not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

LOGICAL OPERATORS

Operator	Description	Example
&&	Logical AND operator. If both the operands are non-zero, then the condition is true.	(A && B) is false.
||	Logical OR Operator. If any of these two operands is non-zero, then condition becomes true.	(A || B) is true.
!	Logical NOT Operator. It is used to reverse the logical state of its operand. If condition is true, then Logical NOT operator will make it false.	!(A && B) is true.

BITWISE OPERATORS

2 & 2

a	b	a & b	a | b	a ^ b
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

OTHER BITWISE OPERATORS

- ~ is the binary one's complement operator
- << is the binary left shift operator
- >> is the binary right shift operator

ASSIGNMENT OPERATORS

Operator	Description
=	Simple assignment operator. Assigns values from right side operands to left side operand
+=	Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand.
-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and the result is assigned to the left operand.
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and the result is assigned to the left operand.
/=	Divide AND assignment operator. It divides the left operand with the right operand and the result is assigned to the left operand.

MISCELLANEOUS OPERATORS

Operator	Description	Example
sizeof()	Returns the size of a variable.	sizeof(a), where a is integer, will return int's size on that architecture.
&	Returns the address of a variable.	&a; returns the actual address of the variable.
*	Pointer to a variable.	*a;
? :	Conditional Expression.	If Condition is true ? then value X : otherwise value Y

OPERATOR PRECEDENCE IN C

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

In programming languages, the **associativity** of an operator is a property that determines how operators of the same precedence are grouped in the absence of parentheses.

```c
#include <stdio.h>
int main() {
    int num, i;
    printf("Enter a Number : ");
    scanf("%d", &num);
    for(i=1;i<=10;i++){
        printf("%d*%d = %d\n", num,i , num*i);
    }
    return 0;
}
```

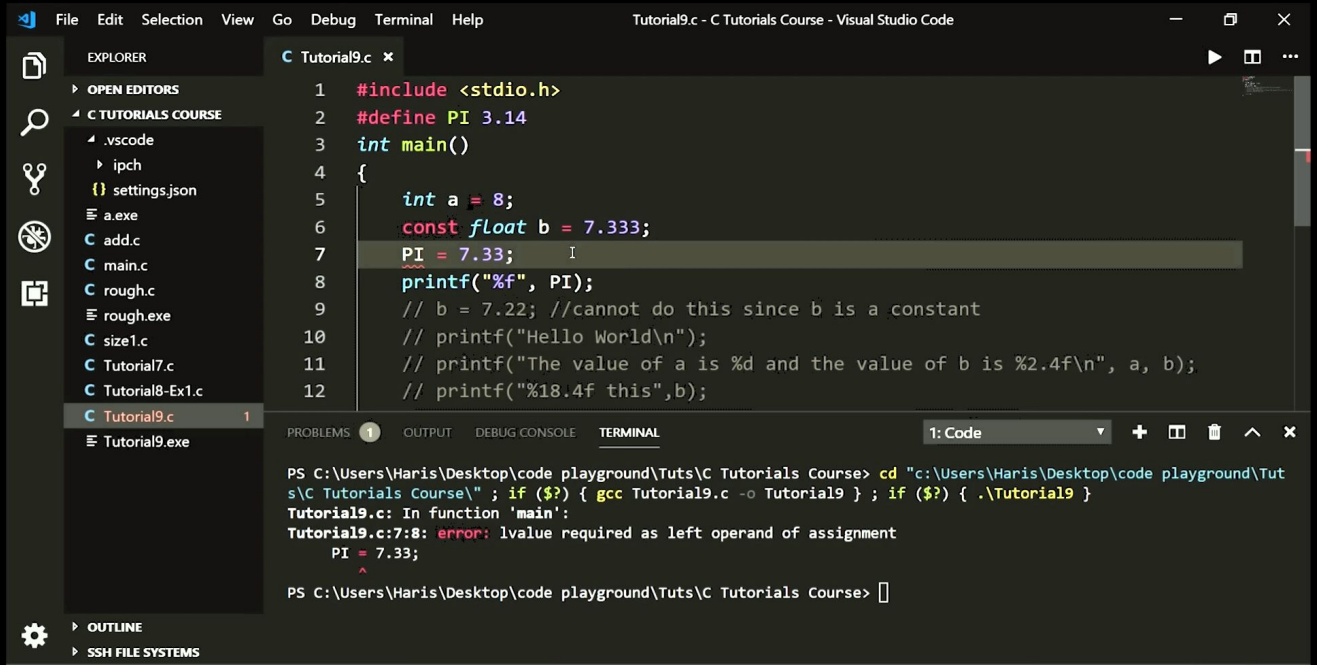
```
/tmp/9v3ZUfVFs6.o
Enter a Number : 5
5*1 = 5
5*2 = 10
5*3 = 15
5*4 = 20
5*5 = 25
5*6 = 30
5*7 = 35
5*8 = 40
5*9 = 45
5*10 = 50
```

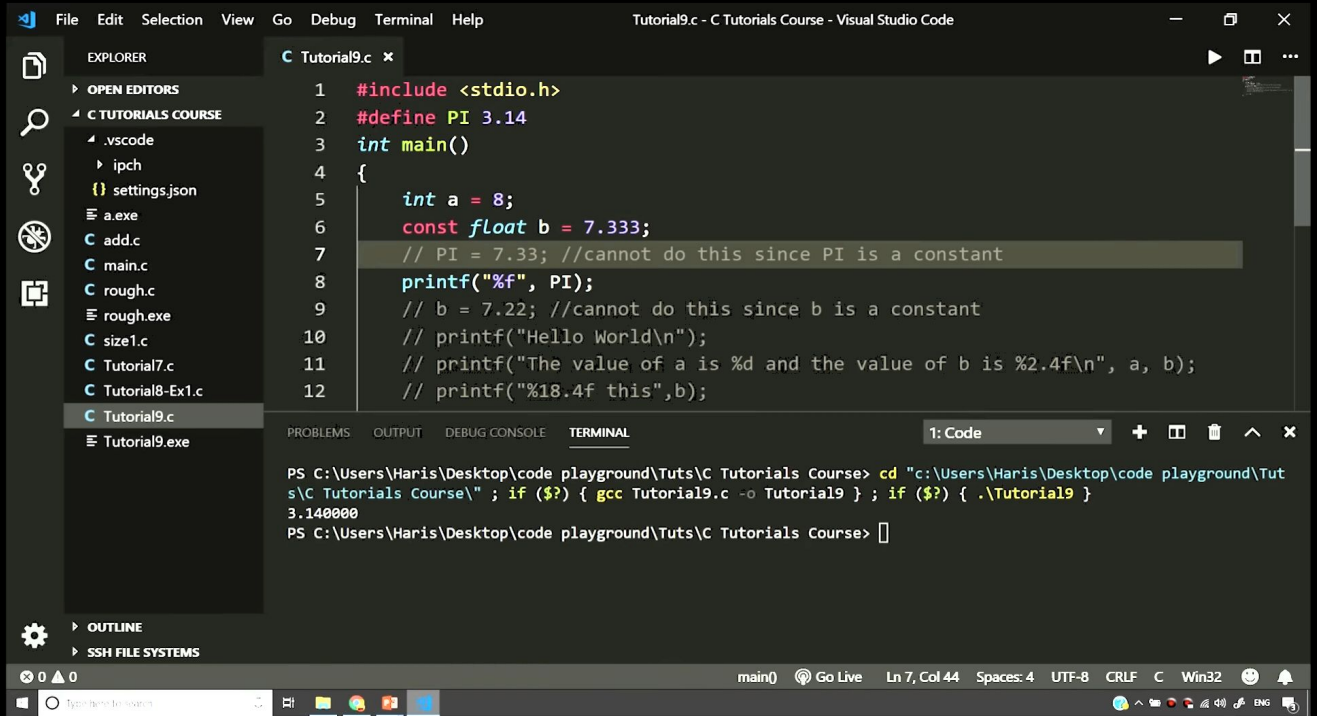
WHAT IS A FORMAT SPECIFIER?

- Format specifier is a way to tell the compiler what type of data is in a variable during taking input displaying output to the user.
- printf("This is a good boy %a.bf", var); will print var with b decimal points in a 'a' character space.
- Lets to to our IDE and learn more about the format specifiers

6 → %Lf → long double

1 → %c → character
2 → %d → Integer
3 → %f → float
4 → %l → long
5 → %lf → double.


File Edit Selection View Go Debug Terminal Help
Tutorial9.c - C Tutorials Course - Visual Studio Code
EXPLORER
OPEN EDITORS
C TUTORIALS COURSE
.vscode
ipch
settings.json
a.exe
add.c
main.c
rough.c
rough.exe
size1.c
Tutorial7.c
Tutorial8-Ex1.c
Tutorial9.c 1
Tutorial9.exe
OUTLINE
SSH FILE SYSTEMS
Tutorial9.c
1 #include <stdio.h>
2 #define PI 3.14
3 int main()
4 {
5 int a = 8;
6 const float b = 7.333;
7 PI = 7.33;
8 printf("%f", PI);
9 // b = 7.22; //cannot do this since b is a constant
10 // printf("Hello World\n");
11 // printf("The value of a is %d and the value of b is %2.4f\n", a, b);
12 // printf("%18.4f this",b);
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL
1: Code
PS C:\Users\Haris\Desktop\code playground\Tuts\C Tutorials Course> cd "c:\Users\Haris\Desktop\code playground\Tuts\C Tutorials Course\" ; if ($?) { gcc Tutorial9.c -o Tutorial9 } ; if ($?) { .\Tutorial9 }
Tutorial9.c: In function 'main':
Tutorial9.c:7:8: error: lvalue required as left operand of assignment
PI = 7.33;
^
PS C:\Users\Haris\Desktop\code playground\Tuts\C Tutorials Course>
1 0
main() Go Live Ln 7, Col 15 Spaces: 4 UTF-8 CRLF C Win32
Subscribe

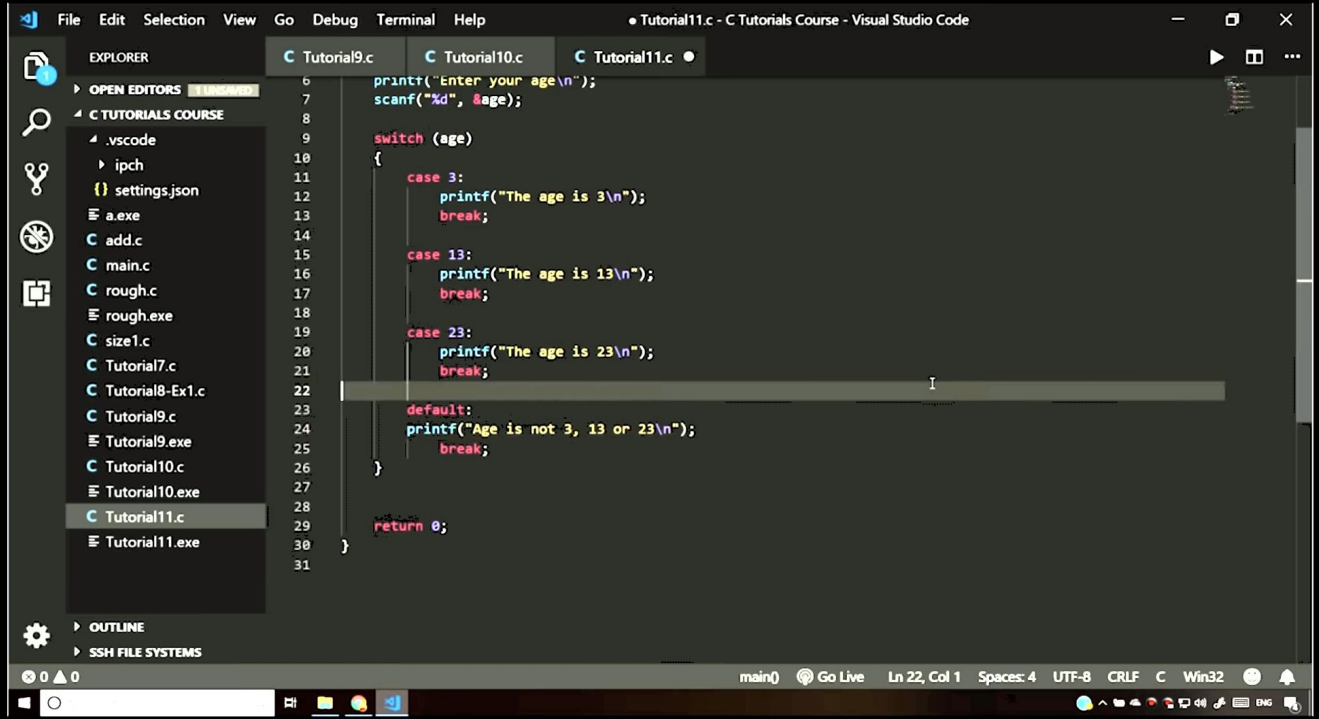

File Edit Selection View Go Debug Terminal Help
Tutorial9.c - C Tutorials Course - Visual Studio Code
EXPLORER
OPEN EDITORS
C TUTORIALS COURSE
.vscode
ipch
settings.json
a.exe
add.c
main.c
rough.c
rough.exe
size1.c
Tutorial7.c
Tutorial8-Ex1.c
Tutorial9.c
Tutorial9.exe
OUTLINE
SSH FILE SYSTEMS
Tutorial9.c
#include <stdio.h>
#define PI 3.14
int main()
{
int a = 8;
const float b = 7.333;
// PI = 7.33; //cannot do this since PI is a constant
printf("%f", PI);
// b = 7.22; //cannot do this since b is a constant
// printf("Hello World\n");
// printf("The value of a is %d and the value of b is %2.4f\n", a, b);
// printf("%18.4f this",b);
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: Code
PS C:\Users\Haris\Desktop\code playground\Tuts\C Tutorials Course> cd "c:\Users\Haris\Desktop\code playground\Tuts\C Tutorials Course\" ; if ($?) { gcc Tutorial9.c -o Tutorial9 } ; if ($?) { .\Tutorial9 }
3.140000
PS C:\Users\Haris\Desktop\code playground\Tuts\C Tutorials Course>
0 0
main() Go Live Ln 7, Col 44 Spaces: 4 UTF-8 CRLF C Win32
Subscribe

C if else statements → Control statements

→ It is used to perform operations based on some condition.

Types of if statements

1> If statement
2> If else statement
3> if -else if ladder
4> Nested if


File Edit Selection View Go Debug Terminal Help
● Tutorial11.c - C Tutorials Course - Visual Studio Code
EXPLORER
OPEN EDITORS 1 UNSAVED
C TUTORIALS COURSE
.vscode
ipch
settings.json
a.exe
add.c
main.c
rough.c
rough.exe
size1.c
Tutorial7.c
Tutorial8-Ex1.c
Tutorial9.c
Tutorial9.exe
Tutorial10.c
Tutorial10.exe
Tutorial11.c
Tutorial11.exe
OUTLINE
SSH FILE SYSTEMS
Tutorial9.c
Tutorial10.c
Tutorial11.c ●
printf("Enter your age\n");
scanf("%d", &age);
switch (age)
{
case 3:
printf("The age is 3\n");
break;
case 13:
printf("The age is 13\n");
break;
case 23:
printf("The age is 23\n");
break;
default:
printf("Age is not 3, 13 or 23\n");
break;
}
return 0;
}
0 0
main() Go Live Ln 22, Col 1 Spaces: 4 UTF-8 CRLF C Win32
Subscribe

Types of Loops

1 → do while loop

2 → while loop

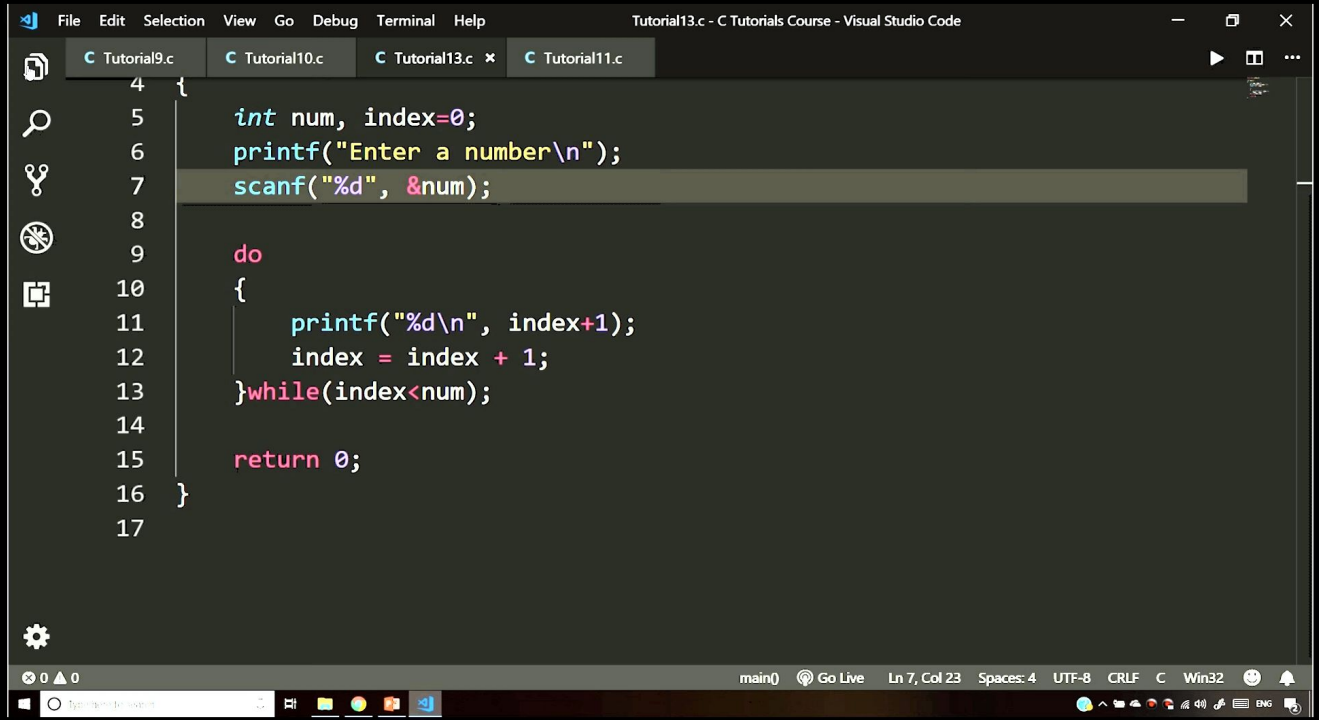
3 → for loop

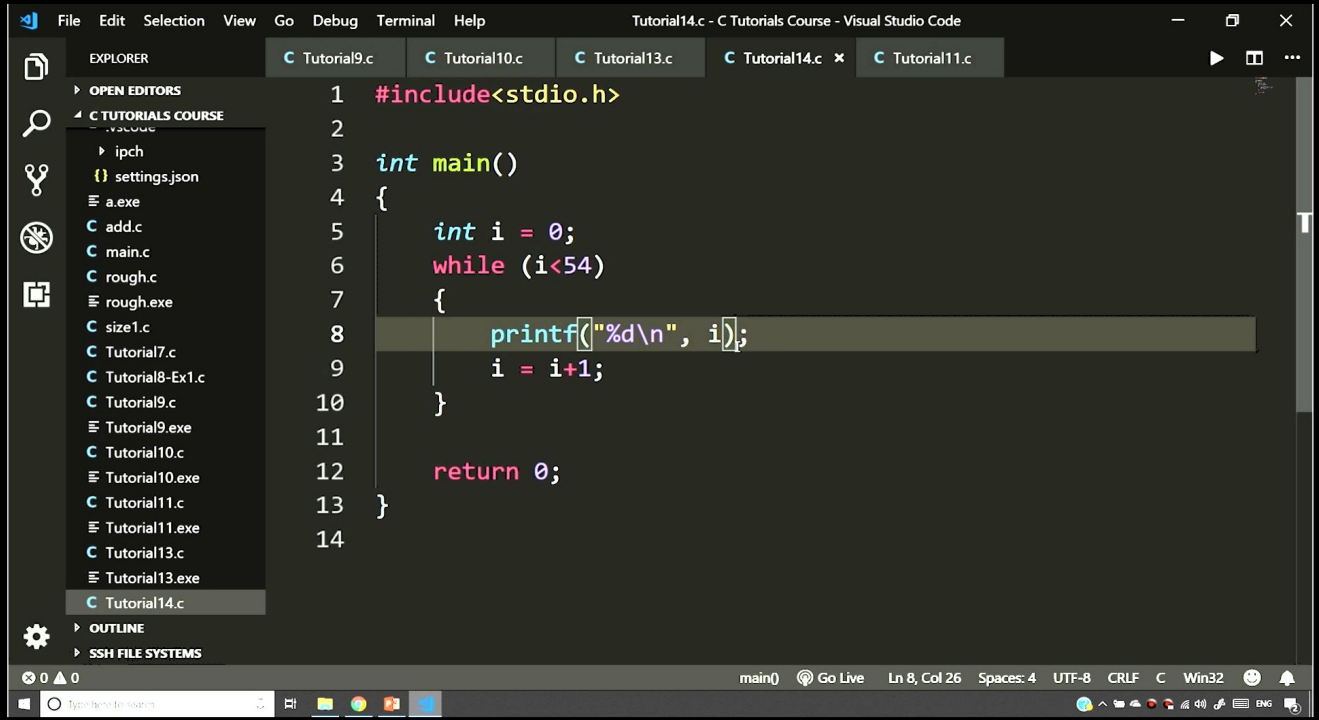
Do While Loop

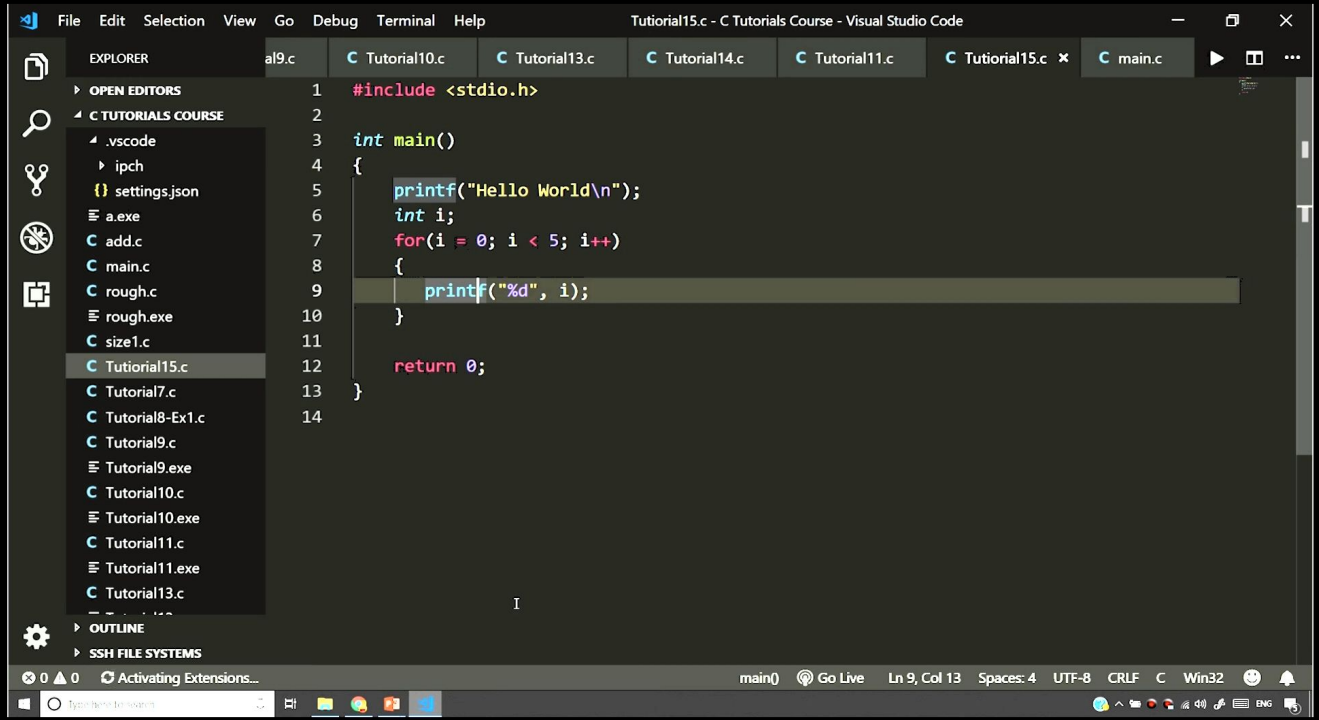
```
do {
    // code to be executed
} while (condition)
```

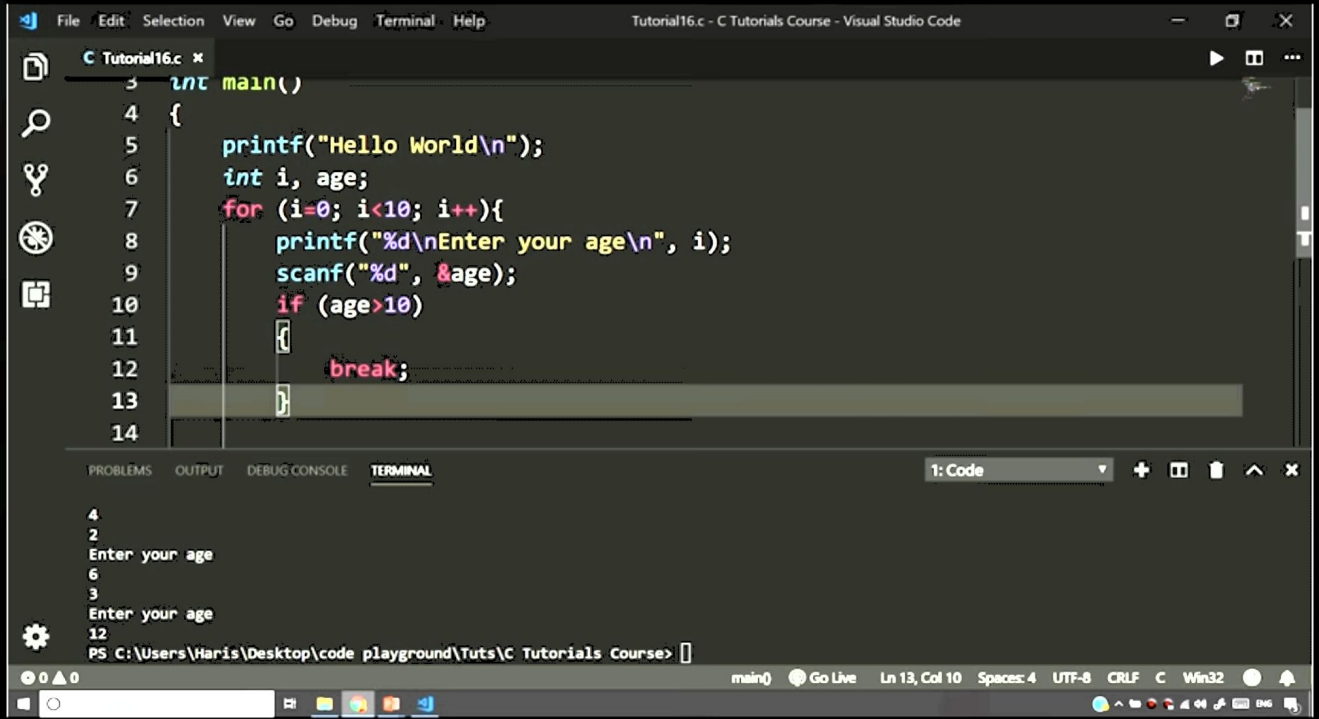
```
int i=0;
do {
    i=i+1;
    printf("%d", i);
} while (i<10);
```

do while loop executes at least once


File Edit Selection View Go Debug Terminal Help
Tutorial13.c - C Tutorials Course - Visual Studio Code
Tutorial9.c
Tutorial10.c
Tutorial13.c
Tutorial11.c
{
int num, index=0;
printf("Enter a number\n");
scanf("%d", &num);
do
{
printf("%d\n", index+1);
index = index + 1;
}while(index<num);
return 0;
}
main()
Go Live
Ln 7, Col 23
Spaces: 4
UTF-8
CRLF
C
Win32
Subscribe


File Edit Selection View Go Debug Terminal Help
Tutorial14.c - C Tutorials Course - Visual Studio Code
EXPLORER
Tutorial9.c
Tutorial10.c
Tutorial13.c
Tutorial14.c
Tutorial11.c
OPEN EDITORS
C TUTORIALS COURSE
ipch
settings.json
a.exe
add.c
main.c
rough.c
rough.exe
size1.c
Tutorial7.c
Tutorial8-Ex1.c
Tutorial9.c
Tutorial9.exe
Tutorial10.c
Tutorial10.exe
Tutorial11.c
Tutorial11.exe
Tutorial13.c
Tutorial13.exe
Tutorial14.c
OUTLINE
SSH FILE SYSTEMS
#include<stdio.h>
int main()
{
int i = 0;
while (i<54)
{
printf("%d\n", i);
i = i+1;
}
return 0;
}
main() Go Live Ln 8, Col 26 Spaces: 4 UTF-8 CRLF C Win32

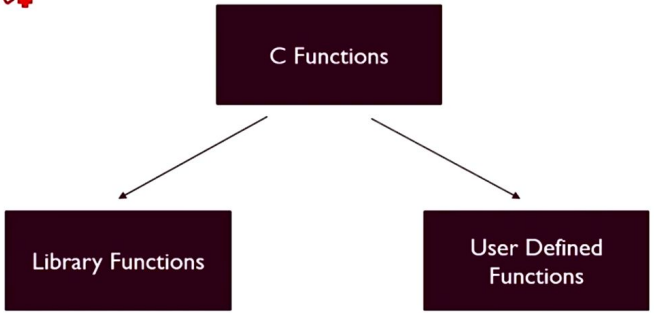

File Edit Selection View Go Debug Terminal Help
Tutorial15.c - C Tutorials Course - Visual Studio Code
EXPLORER
OPEN EDITORS
C TUTORIALS COURSE
.vscode
ipch
settings.json
a.exe
add.c
main.c
rough.c
rough.exe
size1.c
Tutorial15.c
Tutorial7.c
Tutorial8-Ex1.c
Tutorial9.c
Tutorial9.exe
Tutorial10.c
Tutorial10.exe
Tutorial11.c
Tutorial11.exe
Tutorial13.c
OUTLINE
SSH FILE SYSTEMS
Tutorial10.c
Tutorial13.c
Tutorial14.c
Tutorial11.c
Tutorial15.c
main.c
#include <stdio.h>
int main()
{
printf("Hello World\n");
int i;
for(i = 0; i < 5; i++)
{
printf("%d", i);
}
return 0;
}
0 0 Activating Extensions...
main() Go Live Ln 9, Col 13 Spaces: 4 UTF-8 CRLF C Win32
Subscribe


File Edit Selection View Go Debug Terminal Help
Tutorial16.c - C Tutorials Course - Visual Studio Code
C Tutorial16.c
int main()
{
printf("Hello World\n");
int i, age;
for (i=0; i<10; i++){
printf("%d\nEnter your age\n", i);
scanf("%d", &age);
if (age>10)
{
break;
}
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: Code
4
2
Enter your age
6
3
Enter your age
12
PS C:\Users\Haris\Desktop\code playground\Tuts\C Tutorials Course>
main() Go Live Ln 13, Col 10 Spaces: 4 UTF-8 CRLF C Win32

TYPES OF FUNCTIONS

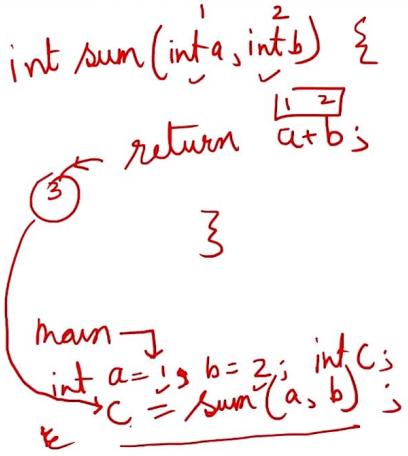
printf(" Li

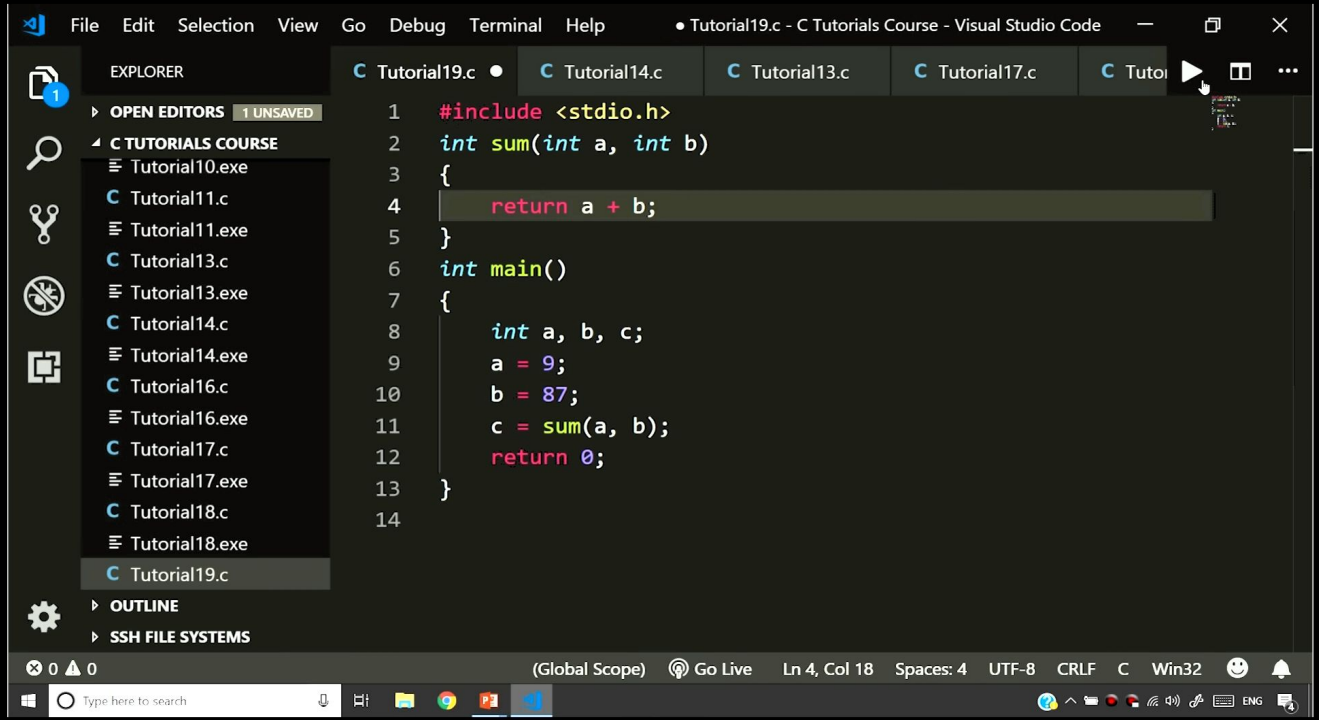
- Library functions – Functions included in c header files
- User defined functions – Functions created by C programmer to reduce complexity of a program



FUNCTION CODE EXAMPLES

- Without arguments and without return value
- Without arguments and with return value
- With arguments and without return value
- With arguments and with return value




File Edit Selection View Go Debug Terminal Help
● Tutorial19.c - C Tutorials Course - Visual Studio Code
EXPLORER
OPEN EDITORS 1 UNSAVED
C TUTORIALS COURSE
Tutorial10.exe
Tutorial11.c
Tutorial11.exe
Tutorial13.c
Tutorial13.exe
Tutorial14.c
Tutorial14.exe
Tutorial16.c
Tutorial16.exe
Tutorial17.c
Tutorial17.exe
Tutorial18.c
Tutorial18.exe
Tutorial19.c
OUTLINE
SSH FILE SYSTEMS
Tutorial19.c ●
Tutorial14.c
Tutorial13.c
Tutorial17.c
1 #include <stdio.h>
2 int sum(int a, int b)
3 {
4 return a + b;
5 }
6 int main()
7 {
8 int a, b, c;
9 a = 9;
10 b = 87;
11 c = sum(a, b);
12 return 0;
13 }
14
0 0
(Global Scope)
Go Live
Ln 4, Col 18
Spaces: 4
UTF-8
CRLF
C
Win32
Type here to search
ENG
Subscribe

```c
// Online C compiler to run C program online
#include <stdio.h>
void sum(){
    int a, b, c;
    printf("Enter the value of a : ");
    scanf("%d", &a);
    printf("Enter the value of b : ");
    scanf("%d", &b);
    c = a+b;
    printf("The sum is %d", c);
}
int main() {
    sum();
    return 0;
}
```

```
/tmp/OBWDsaFKHr.o
Enter the value of a : 5
Enter the value of b : 20
The sum is 25
```

WHAT IS A RECURSIVE FUNCTION?

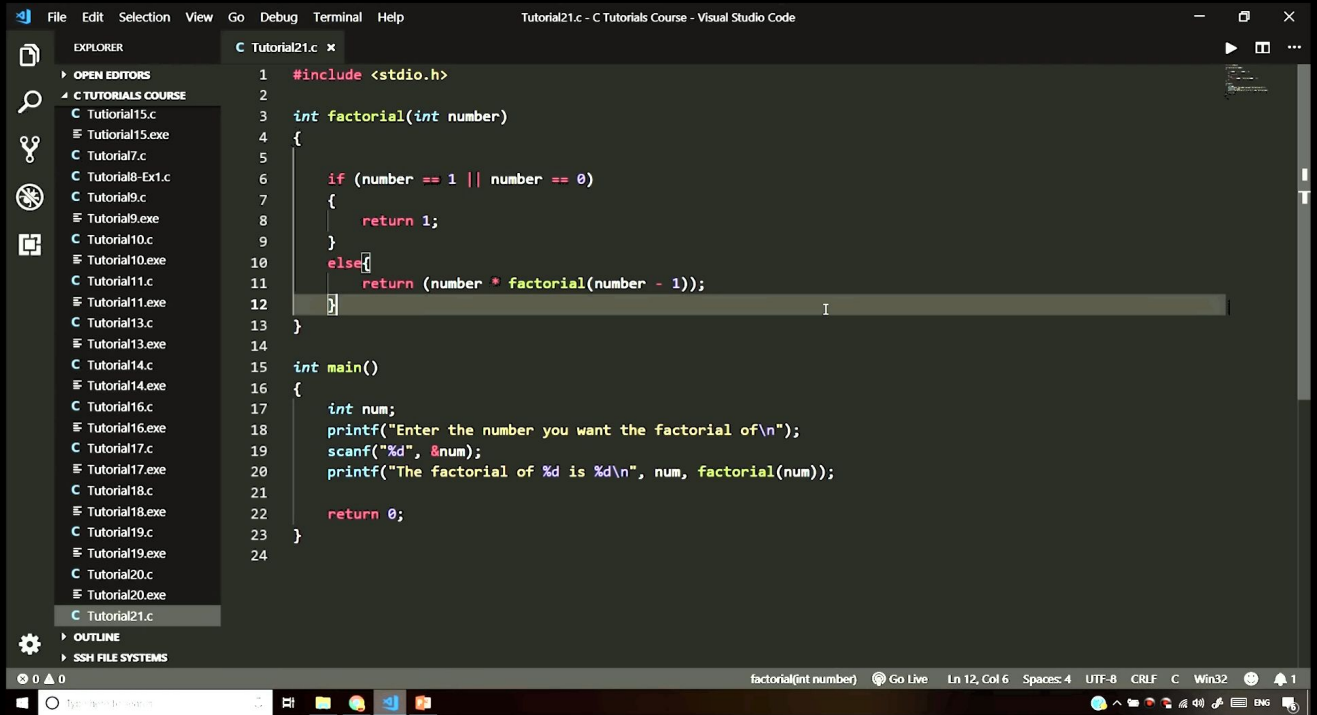
- Recursive functions or Recursion is a process when a function calls a copy of itself to work on a smaller problem.
- Any function which calls itself is called recursive function.
- This makes the life of programmer easy by dividing a given problem into easier problems.
- A termination condition is imposed on such functions to stop them executing copies of themselves forever
- Any problem that can be solved recursively, can also be solved iteratively

WHY RECURSIONS?

- Any problem that can be solved recursively, can also be solved iteratively.
- However, some problems are best suited to be solved using recursion.
- For example, tower of Hanoi, Fibonacci series, factorial finding, etc.

```c
// Online C compiler to run C program online
#include <stdio.h>
int main() {
    int i, n, fact = 1;
    printf("Enter a number : ");
    scanf("%d", &n);
    for(i=1; i<=n; i++){
        fact *=i;
    }
    printf("Factorial of %d is %d", n, fact);
    return 0;
}
```

```
/tmp/8yUcQk9CHO.o
Enter a number : 5
Factorial of 5 is 120
```


File Edit Selection View Go Debug Terminal Help
Tutorial21.c - C Tutorials Course - Visual Studio Code
EXPLORER
OPEN EDITORS
C TUTORIALS COURSE
Tutorial15.c
Tutorial15.exe
Tutorial7.c
Tutorial8-Ex1.c
Tutorial9.c
Tutorial9.exe
Tutorial10.c
Tutorial10.exe
Tutorial11.c
Tutorial11.exe
Tutorial13.c
Tutorial13.exe
Tutorial14.c
Tutorial14.exe
Tutorial16.c
Tutorial16.exe
Tutorial17.c
Tutorial17.exe
Tutorial18.c
Tutorial18.exe
Tutorial19.c
Tutorial19.exe
Tutorial20.c
Tutorial20.exe
Tutorial21.c
OUTLINE
SSH FILE SYSTEMS
Tutorial21.c
#include <stdio.h>
int factorial(int number)
{
if (number == 1 || number == 0)
{
return 1;
}
else{
return (number * factorial(number - 1));
}
}
int main()
{
int num;
printf("Enter the number you want the factorial of\n");
scanf("%d", &num);
printf("The factorial of %d is %d\n", num, factorial(num));
return 0;
}
factorial(int number)
Go Live
Ln 12, Col 6
Spaces: 4
UTF-8
CRLF
C
Win32
Subscribe

Replies

@ishtiaqahmed9829 • 1 yr ago

```
/*Perform The conversion of units
to convert kilometers to miles
 to convert inches to foot
 to convert centimeters to inches
 to convert pounds to kgs
 to convert inches to meters
 By using switch case conditions in C
Language*/
#include <stdio.h>
int main()
{
   int choices;
   float kilometers, miles, inches, foot,
centimeters, pounds, kgs, meters;
   char ch;
   printf("Enter 'Y' if you want to perform
conversions: \n");
   printf("Enter 'N' if you donot want to
perform conversions: \n");
```

Add a reply...

Replies

```
    char ch;
    printf("Enter 'Y' if you want to perform
conversions: \n");
    printf("Enter 'N' if you donot want to
perform conversions: \n");
    scanf(" %c", &ch);
    switch (ch)
    {
    case 'Y':
        printf("Hi, Lets perform the conversions
of units: \n");
        printf("Enter 1 to convert kilometers to
miles: \n");
        printf("Enter 2 to convert inches to foot:
\n");
        printf("Enter 3 to convert centimeters to
inches: \n");
        printf("Enter 4 to convert pounds to kgs:
\n");
        printf("Enter 5 to convert inches to
```

Add a reply...

Replies

```
        printf("Enter 5 to convert inches to
meters : \n");
        scanf("%d", &choices);
        switch (choices)
        {
        case 1:
           printf("Enter kilometers: \n");
           scanf("%f", &kilometers);
           miles = kilometers * 0.621371;
           printf("It is converted to miles at:  %f\n",
miles);
           break;
        case 2:
           printf("Enter inches: \n");
           scanf("%f", &inches);
           foot = inches * 0.08333;
           printf("It is converted to foot at: %f\n",
foot);
           break;
        case 3:
```

Add a reply...

Replies

```
        break;
    case 3:
        printf("Enter centimeters: \n");
        scanf("%f", &centimeters);
        inches = centimeters * 0.393701;
        printf("It is converted to inches at:
%f\n", inches);
        break;
    case 4:
        printf("Enter pounds: \n");
        scanf("%f", &pounds);
        kgs = pounds * 0.453592;
        printf("It is converted to kgs at: %f\n",
kgs);
        break;
    case 5:
        printf("Enter inches: \n");
        scanf("%f", &inches);
        meters = inches * 39.37;
        printf("It is converted to meters at:
%f\n", meters);
```

Add a reply...


```
%f\n",meters);
            break;
        default:
            printf("!!Wrong choice!! \n");
            break;
        }
        main();
        break;
    case 'N':
        printf("You decided not to convert
anything. \n");
        printf("Thanks for visit.\nHave a Nice
Day!\n");
        break;
    default:
        printf("You Entered a Wrong Character!
\n");
        break;
    }
    return 0;
```

WHAT IS AN ARRAY?

- An array is a collection of data items of the same type
- Items are stored at contiguous memory locations.
- It can also store the collection of derived data types, such as pointers, structures, etc.
- A one-dimensional array is like a list.
- A two-dimensional array is like a table.
- The C language places no limits on the number of dimensions in an array.
- Some texts refer to one-dimensional arrays as ***vectors***, two-dimensional arrays as ***matrices***, and use the general term ***arrays*** when the number of dimensions is unspecified or unimportant.

SYNTAX FOR DECLARING AND INITIALIZING AN ARRAY

- Data_type name[size];
- Data_type name[size] = {x, y, z, ……}; //size not required in this case!
- data_type name[rows][columns]; //for 2-d arrays
- We can also initialize the array one by one by accessing it using its index:
  - name[0] = 0;
- Lets go to VS code and write some code.

int marks[200]

marks[0] = 100;
marks[1] = 96;
⋮
marks[199] = 101;

```c
// Online C compiler to run C program online
#include <stdio.h>

int main() {
    int marks[5];
    for(int i=0;i<5;i++){
        printf("Enter the %d element of the array :
            ", i);
        scanf("%d", &marks[i]);
    }
    for(int i=0;i<5;i++){
       printf("The value %d element of the array is
           : %d\n", i, marks[i]);
    }
    return 0;
}
```

```
/tmp/TY431avtfe.o
Enter the 0 element of the array : 10
Enter the 1 element of the array : 20
Enter the 2 element of the array : 30
Enter the 3 element of the array : 40
Enter the 4 element of the array : 50
The value 0 element of the array is : 10
The value 1 element of the array is : 20
The value 2 element of the array is : 30
The value 3 element of the array is : 40
The value 4 element of the array is : 50
```

```c
#include<stdio.h>
int main(){
   int matrix[2][3];
   for(int i=0; i<2; i++) {
      for(int j=0;j<3;j++) {
         printf("Enter value for matrix[%d][%d]:",
             i, j);
         scanf("%d", &matrix[i][j]);
      }
   }

   for(int i=0; i<2; i++) {
      for(int j=0;j<3;j++) {
         printf("%d ", matrix[i][j]);
      }
      printf("\n");
   }
   return 0;
}
```

```
/tmp/YkFHx3P2W2.o
Enter value for matrix[0][0]:1
Enter value for matrix[0][1]:2
Enter value for matrix[0][2]:3
Enter value for matrix[1][0]:4
Enter value for matrix[1][1]:5
Enter value for matrix[1][2]:6
1 2 3
4 5 6
```

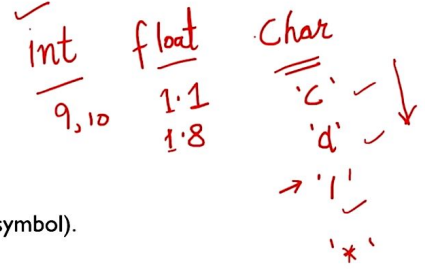
```c
// Online C compiler to run C program online
#include <stdio.h>
int Fibonacci(int n){
    if (n==0){
        return 0;
    }
    else if (n==1){
        return 1;
    }
    else {
        return Fibonacci(n-1) + Fibonacci(n-2);
    }
}

int PrintFibonacci(int n){
    for(int i=0; i<n; i++){
        printf("%d\n", Fibonacci(i));
    }
    return 0;
}

int main() {
    int n;
    printf("Enter till which number you want to
        print Fibonacci sequence : ");
    scanf("%d", &n);
    PrintFibonacci(n);
    return 0;
}
```

```
/tmp/v2k5KYnPND.o
Enter till which number you want to print Fibonacci
    sequence : 8
0
1
1
2
3
5
8
13
```

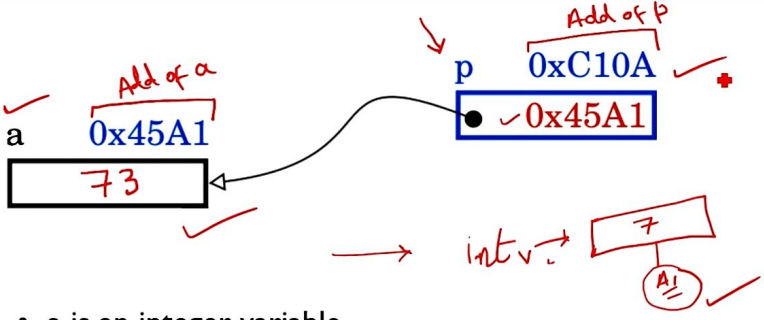
Fibonacci series using recursion

WHAT IS A POINTER?

- Variable which stores the address of another variable.
- Can be of type int, char, array, function, or any other pointer.
- Size depends on the architecture. Ex 2 bytes for 32 bit
- Pointer in C programming language can be declared using * (asterisk symbol).



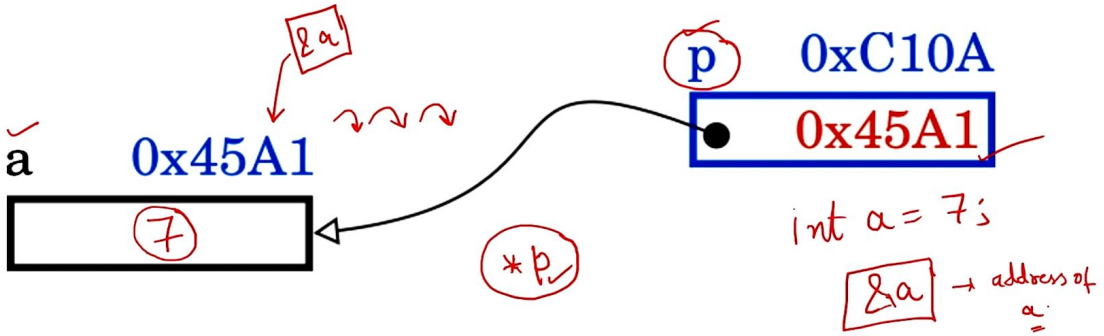
INTUITIVE ANALOGY – STACKED BOXES



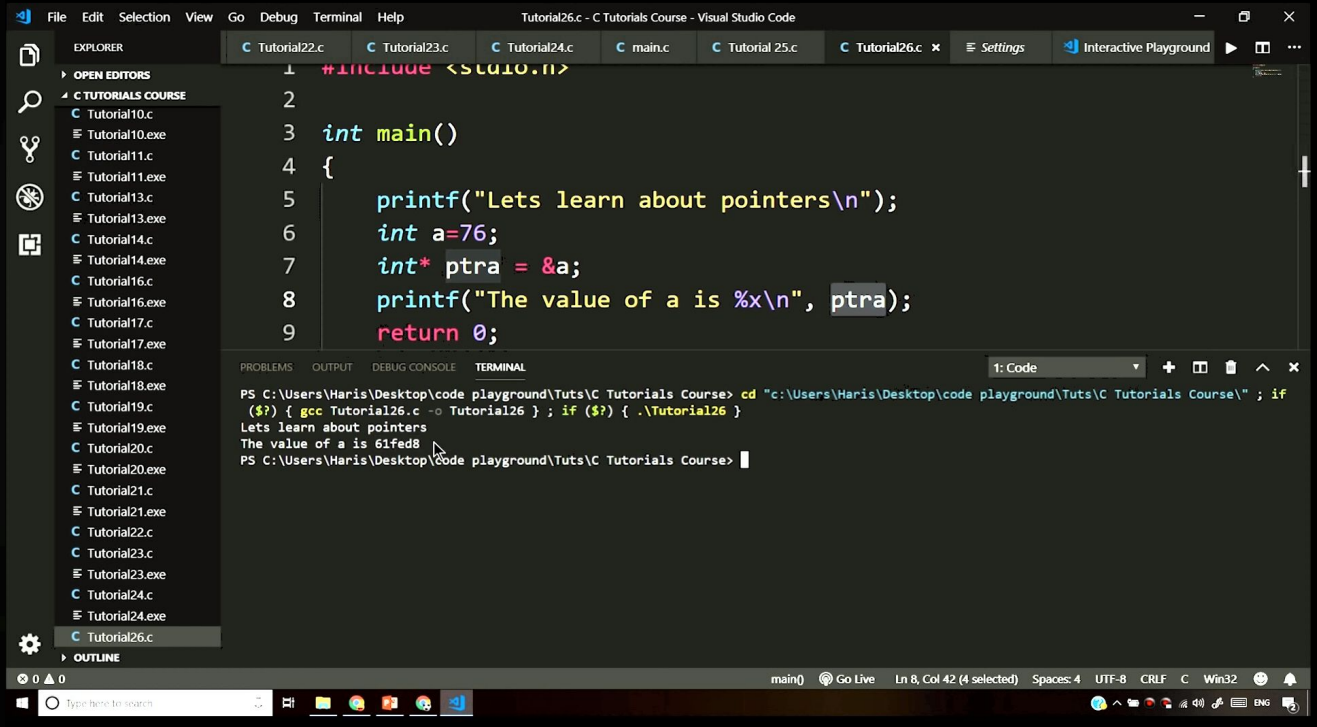
- a is an integer variable
- p is a pointer to integer

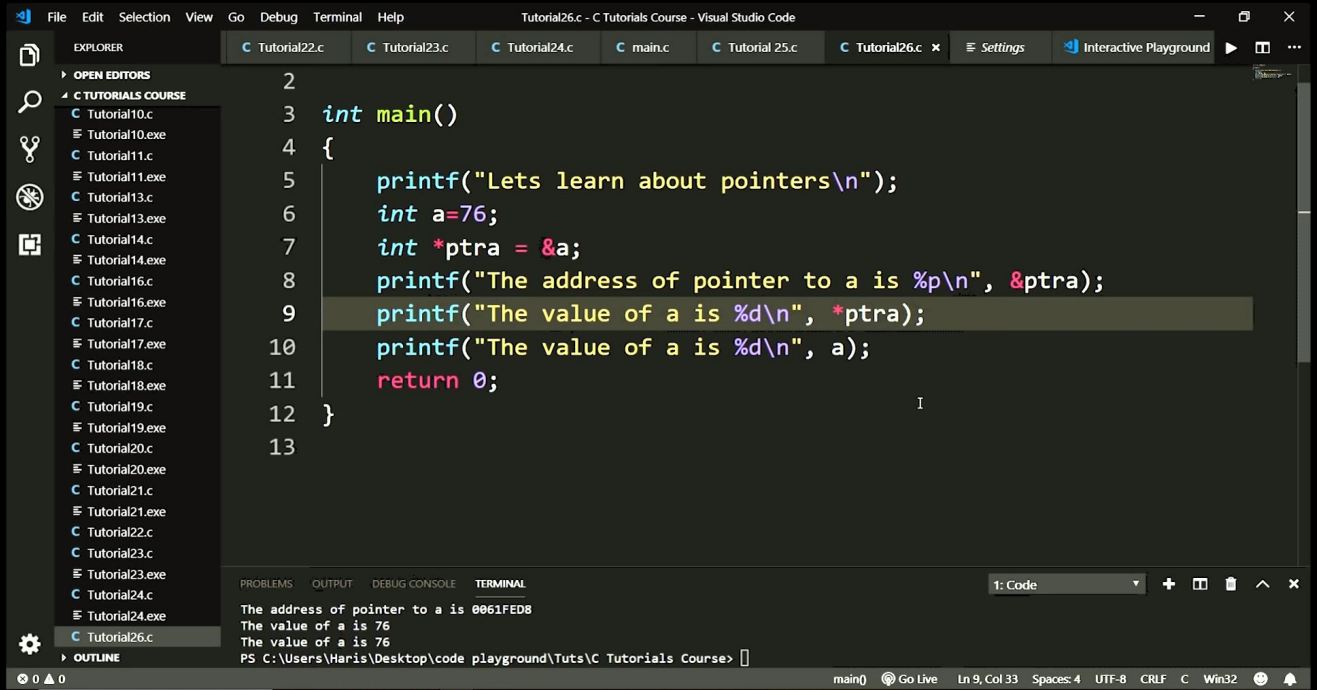
int *ptr = &Var;

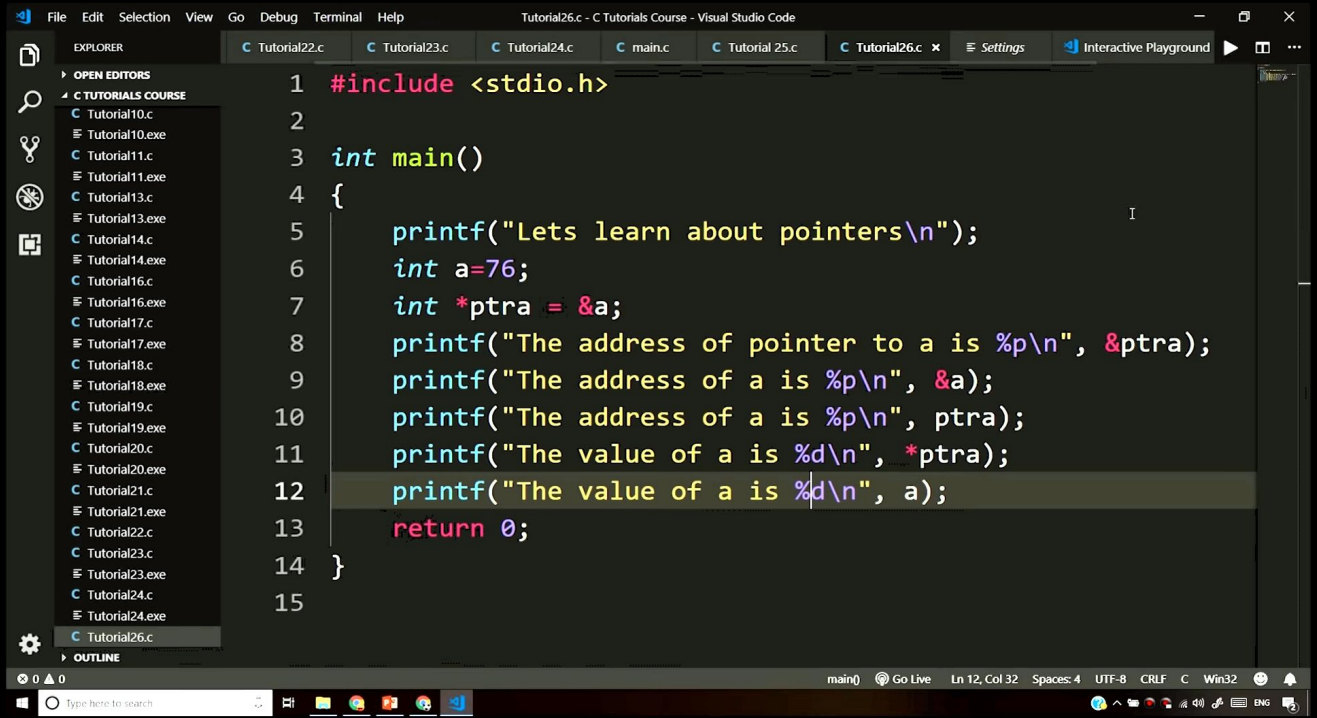
'&' AND '*' OPERATORS

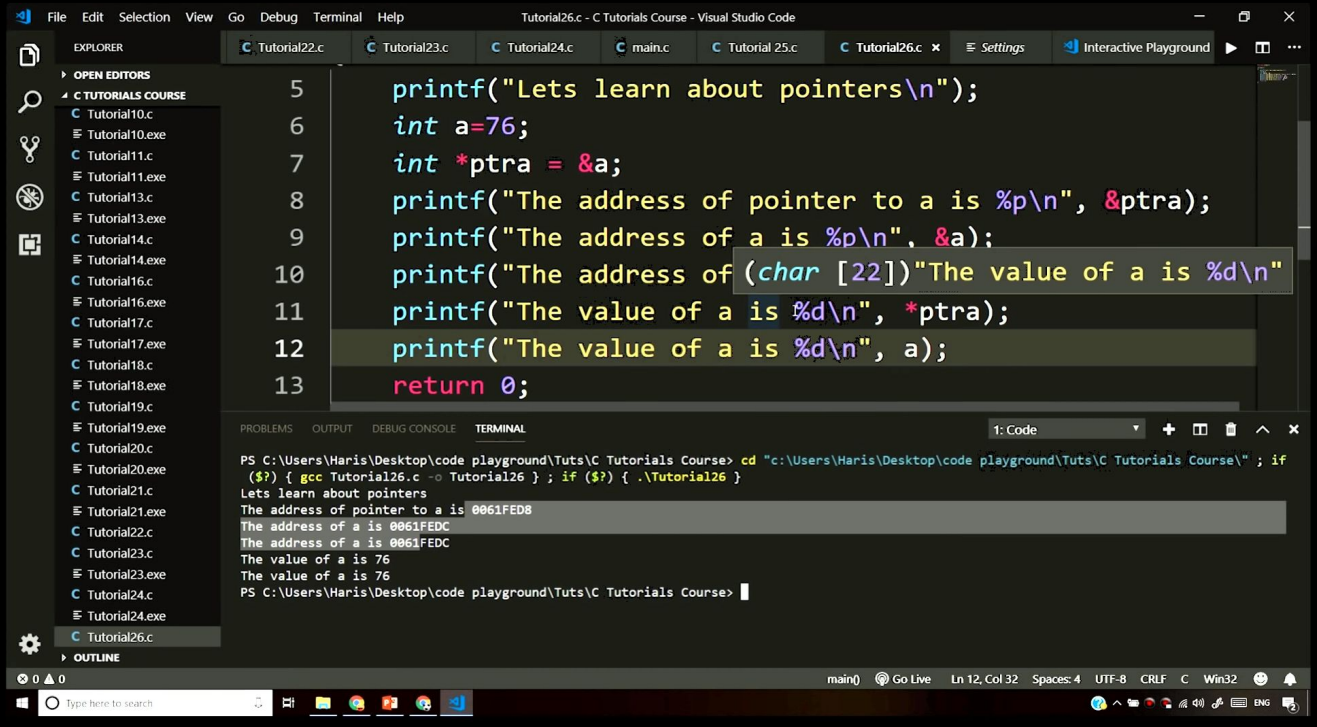


- The address of operator '&' returns the address of a variable
- * is the dereference operator (also called **indirection operator**) used to get the value at a given address


File Edit Selection View Go Debug Terminal Help
Tutorial26.c - C Tutorials Course - Visual Studio Code
EXPLORER
Tutorial22.c
Tutorial23.c
Tutorial24.c
main.c
Tutorial 25.c
Tutorial26.c
Settings
Interactive Playground
OPEN EDITORS
C TUTORIALS COURSE
Tutorial10.c
Tutorial10.exe
Tutorial11.c
Tutorial11.exe
Tutorial13.c
Tutorial13.exe
Tutorial14.c
Tutorial14.exe
Tutorial16.c
Tutorial16.exe
Tutorial17.c
Tutorial17.exe
Tutorial18.c
Tutorial18.exe
Tutorial19.c
Tutorial19.exe
Tutorial20.c
Tutorial20.exe
Tutorial21.c
Tutorial21.exe
Tutorial22.c
Tutorial23.c
Tutorial23.exe
Tutorial24.c
Tutorial24.exe
Tutorial26.c
OUTLINE
#include <stdio.h>
int main()
{
printf("Lets learn about pointers\n");
int a=76;
int* ptra = &a;
printf("The value of a is %x\n", ptra);
return 0;
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
1: Code
PS C:\Users\Haris\Desktop\code playground\Tuts\C Tutorials Course> cd "c:\Users\Haris\Desktop\code playground\Tuts\C Tutorials Course\" ; if ($?) { gcc Tutorial26.c -o Tutorial26 } ; if ($?) { .\Tutorial26 }
Lets learn about pointers
The value of a is 61fed8
PS C:\Users\Haris\Desktop\code playground\Tuts\C Tutorials Course>
main()
Go Live
Ln 8, Col 42 (4 selected)
Spaces: 4
UTF-8
CRLF
C
Win32
Type here to search
ENG
Subscribe


File Edit Selection View Go Debug Terminal Help
Tutorial26.c - C Tutorials Course - Visual Studio Code
EXPLORER
Tutorial22.c
Tutorial23.c
Tutorial24.c
main.c
Tutorial 25.c
Tutorial26.c
Settings
Interactive Playground
OPEN EDITORS
C TUTORIALS COURSE
Tutorial10.c
Tutorial10.exe
Tutorial11.c
Tutorial11.exe
Tutorial13.c
Tutorial13.exe
Tutorial14.c
Tutorial14.exe
Tutorial16.c
Tutorial16.exe
Tutorial17.c
Tutorial17.exe
Tutorial18.c
Tutorial18.exe
Tutorial19.c
Tutorial19.exe
Tutorial20.c
Tutorial20.exe
Tutorial21.c
Tutorial21.exe
Tutorial22.c
Tutorial23.c
Tutorial23.exe
Tutorial24.c
Tutorial24.exe
Tutorial26.c
OUTLINE
2
3 int main()
4 {
5 printf("Lets learn about pointers\n");
6 int a=76;
7 int *ptra = &a;
8 printf("The address of pointer to a is %p\n", &ptra);
9 printf("The value of a is %d\n", *ptra);
10 printf("The value of a is %d\n", a);
11 return 0;
12 }
13
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: Code
The address of pointer to a is 0061FED8
The value of a is 76
The value of a is 76
PS C:\Users\Haris\Desktop\code playground\Tuts\C Tutorials Course>
0 0
main() Go Live Ln 9, Col 33 Spaces: 4 UTF-8 CRLF C Win32
Type here to search
ENG
Subscribe


File Edit Selection View Go Debug Terminal Help
Tutorial26.c - C Tutorials Course - Visual Studio Code
EXPLORER
OPEN EDITORS
C TUTORIALS COURSE
Tutorial10.c
Tutorial10.exe
Tutorial11.c
Tutorial11.exe
Tutorial13.c
Tutorial13.exe
Tutorial14.c
Tutorial14.exe
Tutorial16.c
Tutorial16.exe
Tutorial17.c
Tutorial17.exe
Tutorial18.c
Tutorial18.exe
Tutorial19.c
Tutorial19.exe
Tutorial20.c
Tutorial20.exe
Tutorial21.c
Tutorial21.exe
Tutorial22.c
Tutorial23.c
Tutorial23.exe
Tutorial24.c
Tutorial24.exe
Tutorial26.c
OUTLINE
Tutorial22.c
Tutorial23.c
Tutorial24.c
main.c
Tutorial 25.c
Tutorial26.c
Settings
Interactive Playground
1 #include <stdio.h>
2
3 int main()
4 {
5 printf("Lets learn about pointers\n");
6 int a=76;
7 int *ptra = &a;
8 printf("The address of pointer to a is %p\n", &ptra);
9 printf("The address of a is %p\n", &a);
10 printf("The address of a is %p\n", ptra);
11 printf("The value of a is %d\n", *ptra);
12 printf("The value of a is %d\n", a);
13 return 0;
14 }
15
main() Go Live Ln 12, Col 32 Spaces: 4 UTF-8 CRLF C Win32
Type here to search
ENG
Subscribe


File Edit Selection View Go Debug Terminal Help
Tutorial26.c - C Tutorials Course - Visual Studio Code
EXPLORER
Tutorial22.c
Tutorial23.c
Tutorial24.c
main.c
Tutorial 25.c
Tutorial26.c
Settings
Interactive Playground
OPEN EDITORS
C TUTORIALS COURSE
Tutorial10.c
Tutorial10.exe
Tutorial11.c
Tutorial11.exe
Tutorial13.c
Tutorial13.exe
Tutorial14.c
Tutorial14.exe
Tutorial16.c
Tutorial16.exe
Tutorial17.c
Tutorial17.exe
Tutorial18.c
Tutorial18.exe
Tutorial19.c
Tutorial19.exe
Tutorial20.c
Tutorial20.exe
Tutorial21.c
Tutorial21.exe
Tutorial22.c
Tutorial23.c
Tutorial23.exe
Tutorial24.c
Tutorial24.exe
Tutorial26.c
OUTLINE
5 printf("Lets learn about pointers\n");
6 int a=76;
7 int *ptra = &a;
8 printf("The address of pointer to a is %p\n", &ptra);
9 printf("The address of a is %p\n", &a);
10 printf("The address of (char [22])"The value of a is %d\n"
11 printf("The value of a is %d\n", *ptra);
12 printf("The value of a is %d\n", a);
13 return 0;
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: Code
PS C:\Users\Haris\Desktop\code playground\Tuts\C Tutorials Course> cd "c:\Users\Haris\Desktop\code playground\Tuts\C Tutorials Course\" ; if ($?) { gcc Tutorial26.c -o Tutorial26 } ; if ($?) { .\Tutorial26 }
Lets learn about pointers
The address of pointer to a is 0061FED8
The address of a is 0061FEDC
The address of a is 0061FEDC
The value of a is 76
The value of a is 76
PS C:\Users\Haris\Desktop\code playground\Tuts\C Tutorials Course>
0 0
main()
Go Live
Ln 12, Col 32
Spaces: 4
UTF-8
CRLF
C
Win32
Type here to search
ENG
Subscribe

```c
// Online C compiler to run C program online
#include <stdio.h>

int main() {
    int a = 5;
    int *ptr = &a;
    printf("%d\n", a);
    printf("%d\n", *ptr);
    printf("%d\n", &a);
    printf("%d\n", ptr);
    printf("%d\n", &ptr);


    return 0;
}
```

```
/tmp/jJ81C
5
5
49308668
49308668
49308656
```

NULL POINTER

- A pointer that is not assigned any value but NULL is known as the NULL pointer.
- In computer programming, a null pointer is a pointer that does not point to any object or function.
- We can use it to initialize a pointer variable when that pointer variable isn't assigned any valid memory address yet.
- int * ptr = NULL;

POINTER ARITHMETIC

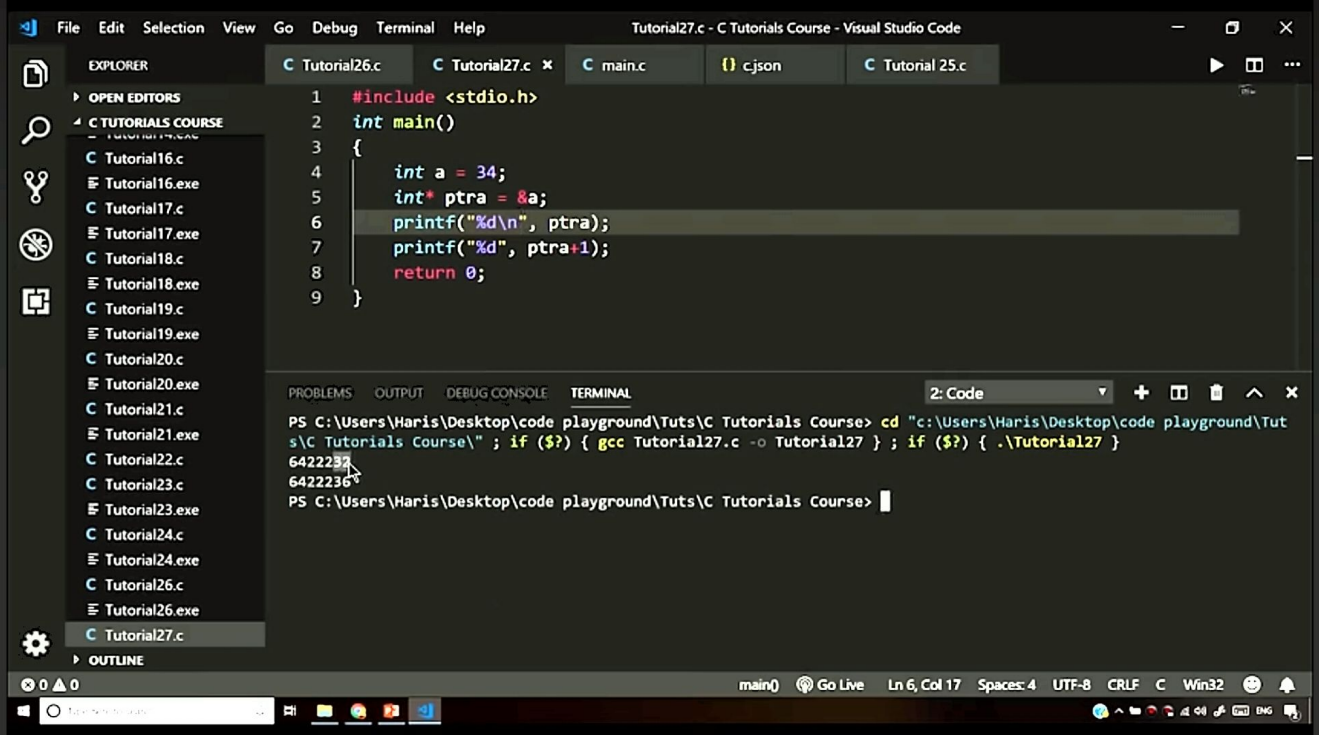
- There are four arithmetic operators that can be used on pointers:
  - ++
  - --
  - +
  - -
- Lets go to VS Code to write some code & learn more!

int a = 2;
a++; → 3

int* ptra = &a;

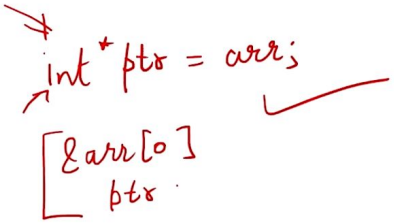
ptra = ptra + 1;

Sizeof(int)

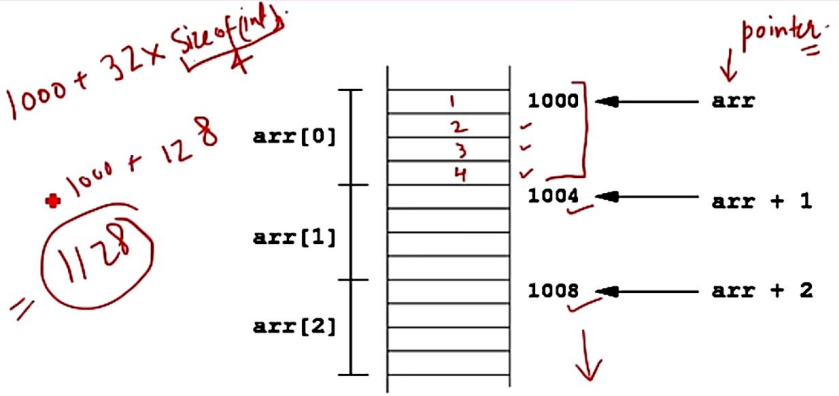

File Edit Selection View Go Debug Terminal Help
Tutorial27.c - C Tutorials Course - Visual Studio Code
EXPLORER
OPEN EDITORS
C TUTORIALS COURSE
Tutorial16.c
Tutorial16.exe
Tutorial17.c
Tutorial17.exe
Tutorial18.c
Tutorial18.exe
Tutorial19.c
Tutorial19.exe
Tutorial20.c
Tutorial20.exe
Tutorial21.c
Tutorial21.exe
Tutorial22.c
Tutorial23.c
Tutorial23.exe
Tutorial24.c
Tutorial24.exe
Tutorial26.c
Tutorial26.exe
Tutorial27.c
OUTLINE
Tutorial26.c
Tutorial27.c
main.c
c.json
Tutorial 25.c
#include <stdio.h>
int main()
{
int a = 34;
int* ptra = &a;
printf("%d\n", ptra);
printf("%d", ptra+1);
return 0;
}
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
2: Code
PS C:\Users\Haris\Desktop\code playground\Tuts\C Tutorials Course> cd "c:\Users\Haris\Desktop\code playground\Tuts\C Tutorials Course\" ; if ($?) { gcc Tutorial27.c -o Tutorial27 } ; if ($?) { .\Tutorial27 }
6422232
6422236
PS C:\Users\Haris\Desktop\code playground\Tuts\C Tutorials Course>
main() Go Live Ln 6, Col 17 Spaces: 4 UTF-8 CRLF C Win32
Subscribe

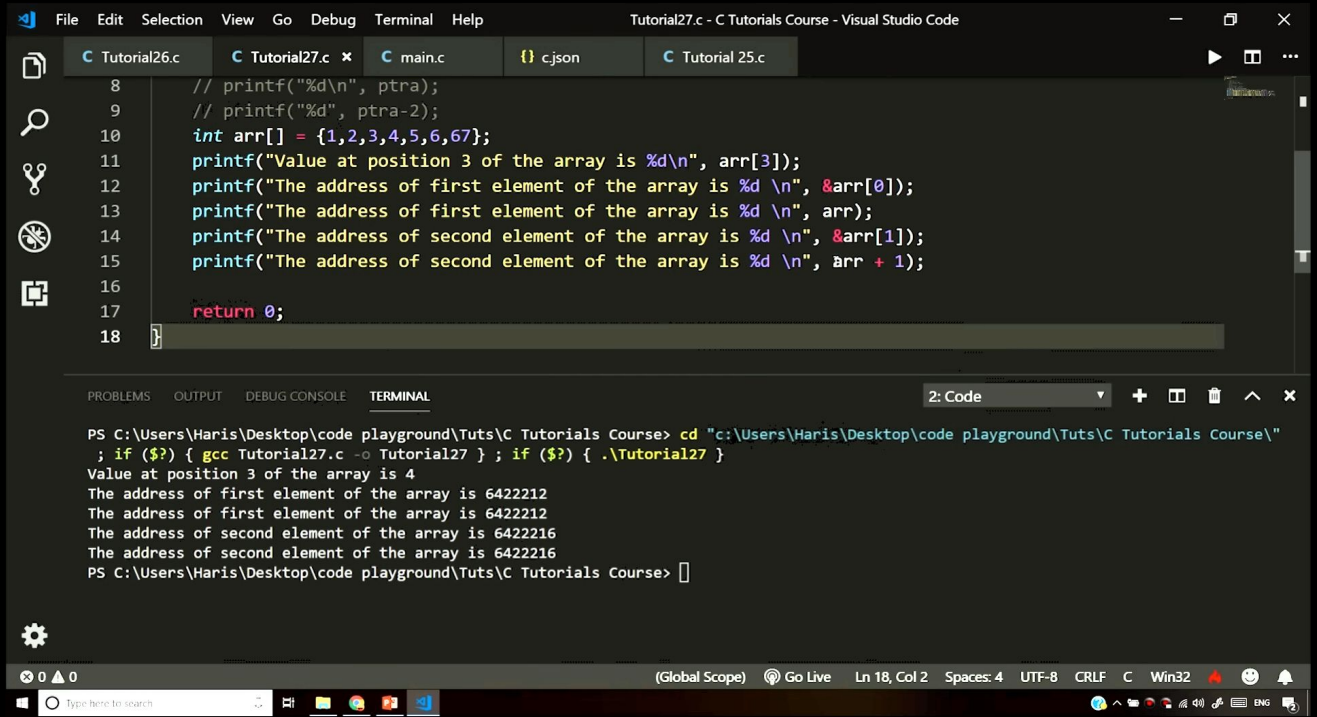
ARRAYS AND POINTERS

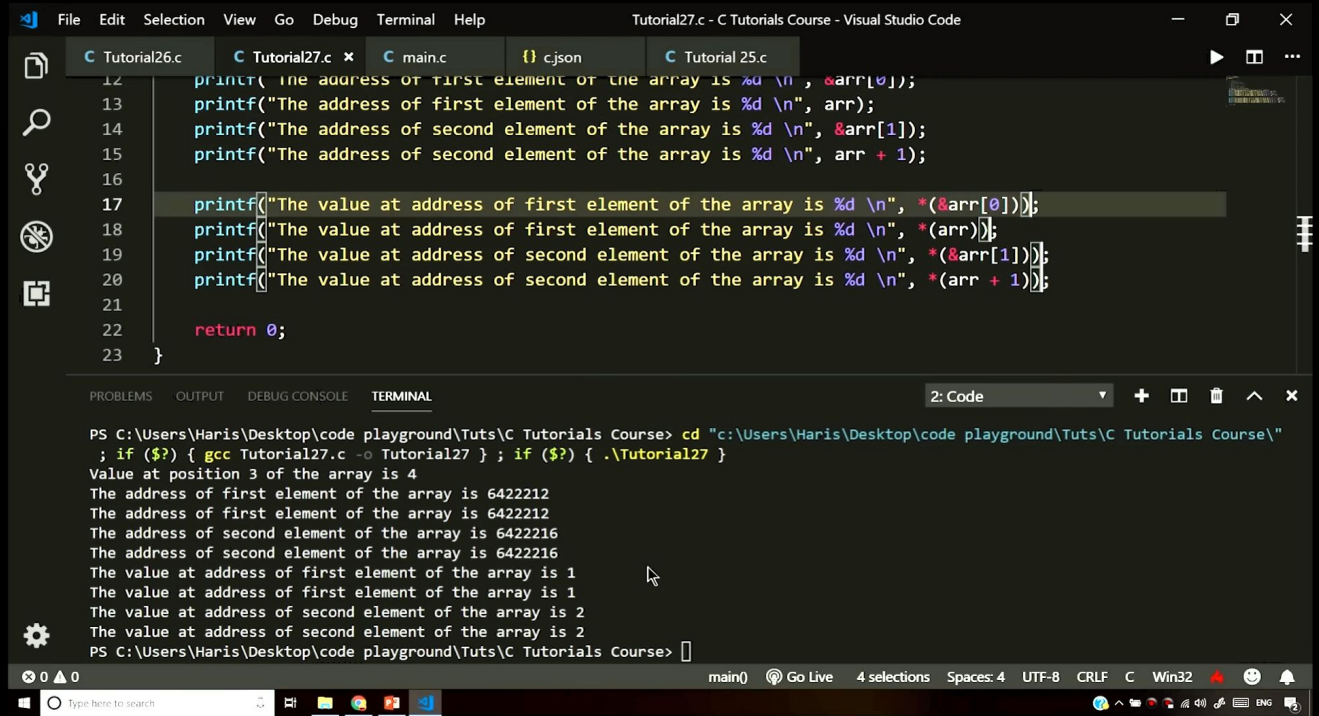
- Consider the following declaration: int arr[10] ;
- What is the type of arr? integer array
- However, **arr**, by itself, without any index subscripting, can be assigned to an integer pointer.
- What type does **arr[i]** have?



ARRAYS IN MEMORY

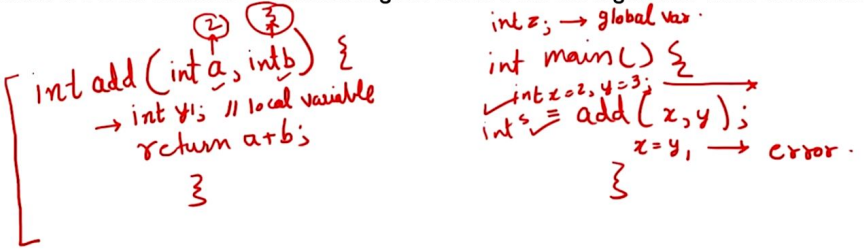



File Edit Selection View Go Debug Terminal Help
Tutorial27.c - C Tutorials Course - Visual Studio Code
Tutorial26.c
Tutorial27.c
main.c
c.json
Tutorial 25.c
// printf("%d\n", ptra);
// printf("%d", ptra-2);
int arr[] = {1,2,3,4,5,6,67};
printf("Value at position 3 of the array is %d\n", arr[3]);
printf("The address of first element of the array is %d \n", &arr[0]);
printf("The address of first element of the array is %d \n", arr);
printf("The address of second element of the array is %d \n", &arr[1]);
printf("The address of second element of the array is %d \n", arr + 1);
return 0;
}
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
2: Code
PS C:\Users\Haris\Desktop\code playground\Tuts\C Tutorials Course> cd "c:\Users\Haris\Desktop\code playground\Tuts\C Tutorials Course\"
; if ($?) { gcc Tutorial27.c -o Tutorial27 } ; if ($?) { .\Tutorial27 }
Value at position 3 of the array is 4
The address of first element of the array is 6422212
The address of first element of the array is 6422212
The address of second element of the array is 6422216
The address of second element of the array is 6422216
PS C:\Users\Haris\Desktop\code playground\Tuts\C Tutorials Course>
0 0
(Global Scope) Go Live Ln 18, Col 2 Spaces: 4 UTF-8 CRLF C Win32
Type here to search
ENG
Subscribe


File Edit Selection View Go Debug Terminal Help
Tutorial27.c - C Tutorials Course - Visual Studio Code
Tutorial26.c
Tutorial27.c
main.c
c.json
Tutorial 25.c
printf("The address of first element of the array is %d \n", &arr[0]);
printf("The address of first element of the array is %d \n", arr);
printf("The address of second element of the array is %d \n", &arr[1]);
printf("The address of second element of the array is %d \n", arr + 1);
printf("The value at address of first element of the array is %d \n", *(&arr[0]));
printf("The value at address of first element of the array is %d \n", *(arr));
printf("The value at address of second element of the array is %d \n", *(&arr[1]));
printf("The value at address of second element of the array is %d \n", *(arr + 1));
return 0;
}
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
2: Code
PS C:\Users\Haris\Desktop\code playground\Tuts\C Tutorials Course> cd "c:\Users\Haris\Desktop\code playground\Tuts\C Tutorials Course\" ; if ($?) { gcc Tutorial27.c -o Tutorial27 } ; if ($?) { .\Tutorial27 }
Value at position 3 of the array is 4
The address of first element of the array is 6422212
The address of first element of the array is 6422212
The address of second element of the array is 6422216
The address of second element of the array is 6422216
The value at address of first element of the array is 1
The value at address of first element of the array is 1
The value at address of second element of the array is 2
The value at address of second element of the array is 2
PS C:\Users\Haris\Desktop\code playground\Tuts\C Tutorials Course>
0 0
main() Go Live 4 selections Spaces: 4 UTF-8 CRLF C Win32
Type here to search
ENG
Subscribe

ACTUAL AND FORMAL PARAMETERS

- When a function is *called*, the values (expressions) that are passed in the call are called the *arguments* or *actual parameters*. [x, y]
- *Formal parameters* are local variables which are assigned values from the arguments when the function is called.



TYPES OF FUNCTION CALLS IN C PROGRAMMING

- In C programming language, we can call a function in two different ways, based on how we specify the arguments, and these two ways are:
  - Call by Value
  - Call by Reference

```
func1(int *a)
{
*a
}
```

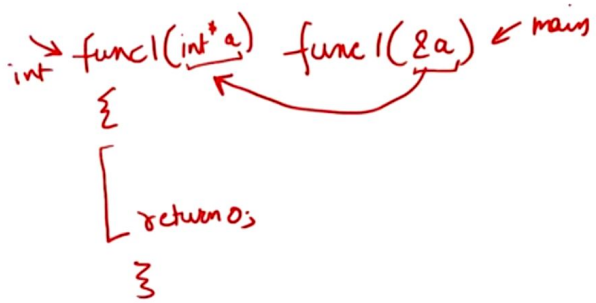
```
main() {
int x=7;
 func1(&x)
  func1(&x);
}
```

CALL BY VALUE

- When we call a function by value, it means that we are passing the values of the arguments which are copied into the formal parameters of the function.
- Which means that the original values remain unchanged and only the parameters inside the function changes.

CALL BY REFERENCE

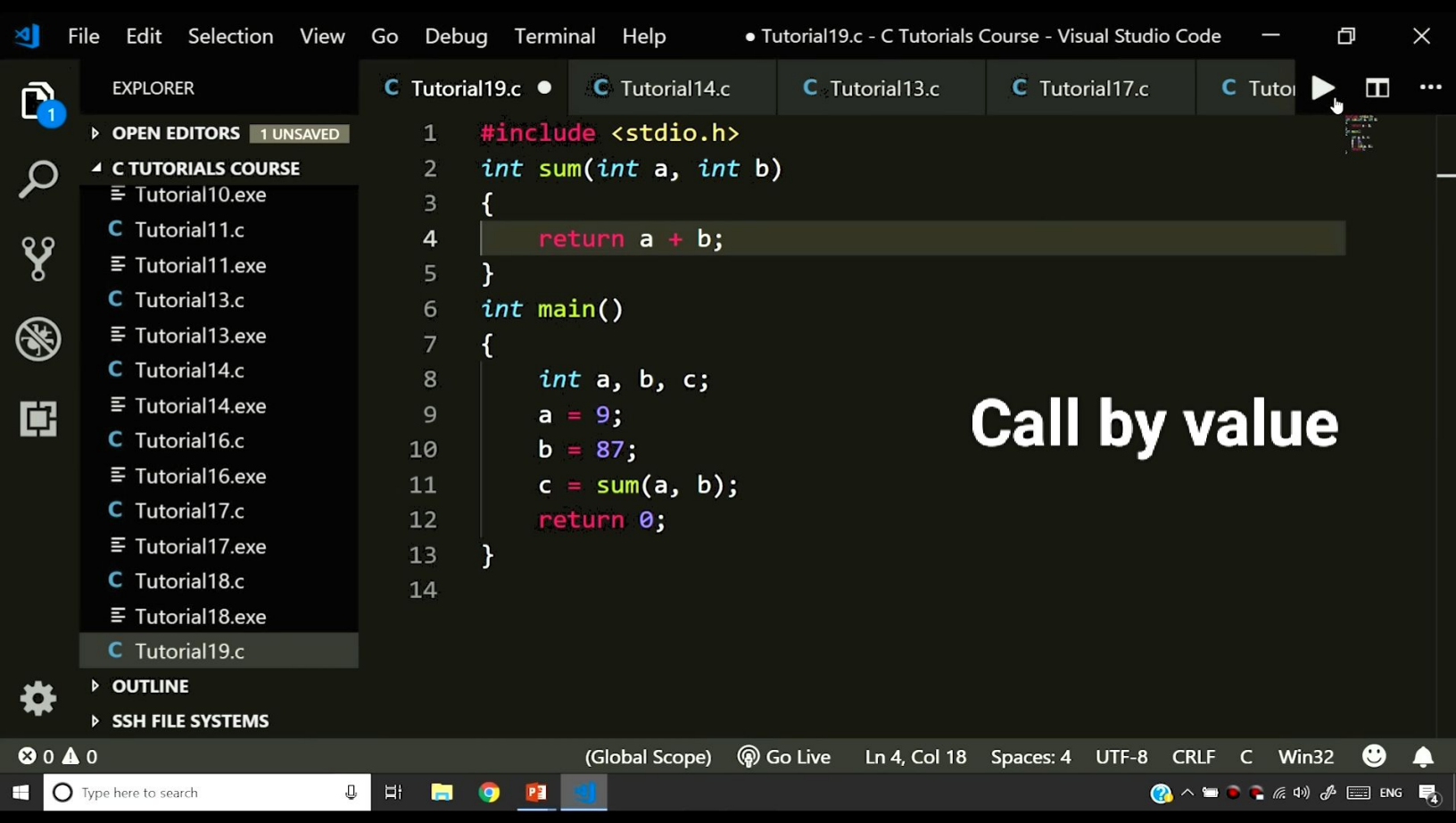
- The **call by reference** method of passing arguments to a C function copies the address of the arguments into the formal parameters
- Addresses of the actual arguments are copied and then assigned to the corresponding formal arguments



CALL BY REFERENCE: EXAMPLE

```
#include <stdio.h>
/* function definition to swap the values */
void swap(int *x, int *y) {

int temp;
temp = *x; /* save the value at address x */
*x = *y; /* put y into x */
*y = temp; /* put temp into y */
return;
}
int main()
{
int a=34, b=74;
printf("%d and %d\n",a, b);
swap(&a, &b);
printf("%d and %d\n",a, b);
return 0;
}
```

34 and 74
74 and 34.


File Edit Selection View Go Debug Terminal Help
• Tutorial19.c - C Tutorials Course - Visual Studio Code
EXPLORER
Tutorial19.c
Tutorial14.c
Tutorial13.c
Tutorial17.c
OPEN EDITORS 1 UNSAVED
C TUTORIALS COURSE
Tutorial10.exe
Tutorial11.c
Tutorial11.exe
Tutorial13.c
Tutorial13.exe
Tutorial14.c
Tutorial14.exe
Tutorial16.c
Tutorial16.exe
Tutorial17.c
Tutorial17.exe
Tutorial18.c
Tutorial18.exe
Tutorial19.c
OUTLINE
SSH FILE SYSTEMS
#include <stdio.h>
int sum(int a, int b)
{
return a + b;
}
int main()
{
int a, b, c;
a = 9;
b = 87;
c = sum(a, b);
return 0;
}
Call by value
0 0
(Global Scope)
Go Live
Ln 4, Col 18
Spaces: 4
UTF-8
CRLF
C
Win32
Type here to search
ENG

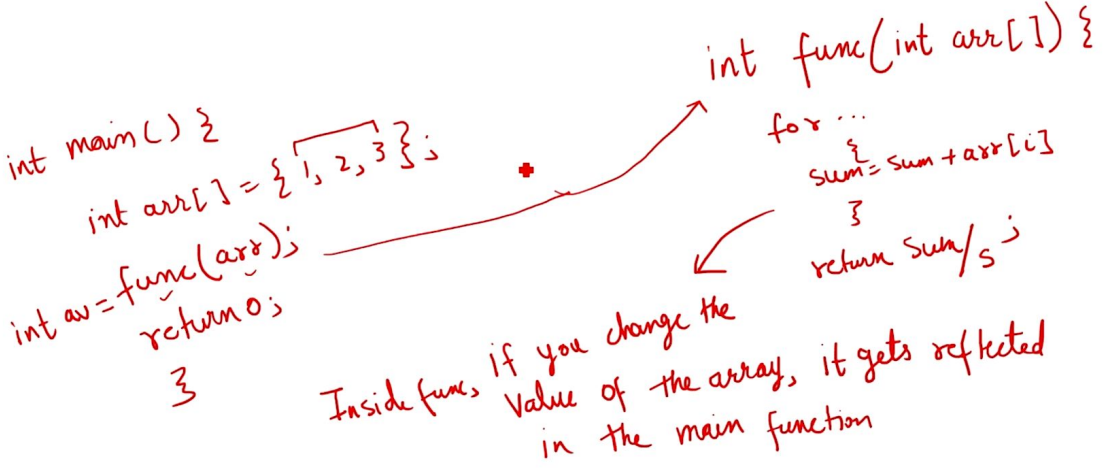
WHY AND HOW TO PASS ARRAYS?

70 Students → Marks.

func1(int n, int n2, ...)

- We pass arrays to a function when we need to pass a list of values to a given function.
- We can pass the arrays to a function :
  1. By declaring array as a parameter in the function → int avg(int arr[])
  2. By declaring a pointer in the function to hold the base address of the array → int sum(int* ptr)

BY DECLARING ARRAY AS A PARAMETER IN THE FUNCTION



```c
// Online C compiler to run C program online
#include <stdio.h>
int avg(int arr[]){
   int sum = 0;

   for(int i=0; i<5; i++){
       sum += arr[i];
   }

   printf("The sum of all the numbers is :
       %d\n", sum);
   printf("The average of these numbers is :
       %d\n", sum/5);
}
int main() {
    int arr[] = {1, 2, 3, 4, 5};
    avg(arr);

    return 0;
}
```

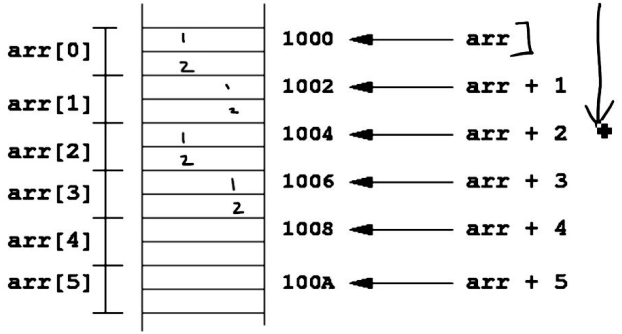
```
/tmp/ZeLOnZ5WTB.o
The sum of all the numbers is : 15
The average of these numbers is : 3
```

BY PASSING ARRAY'S BASE ADDRESS TO THE FUNCTION

```
int main ()
{
  int arr[] = {1,2,3}
  avg = func(arr)

  return 0;
}
```

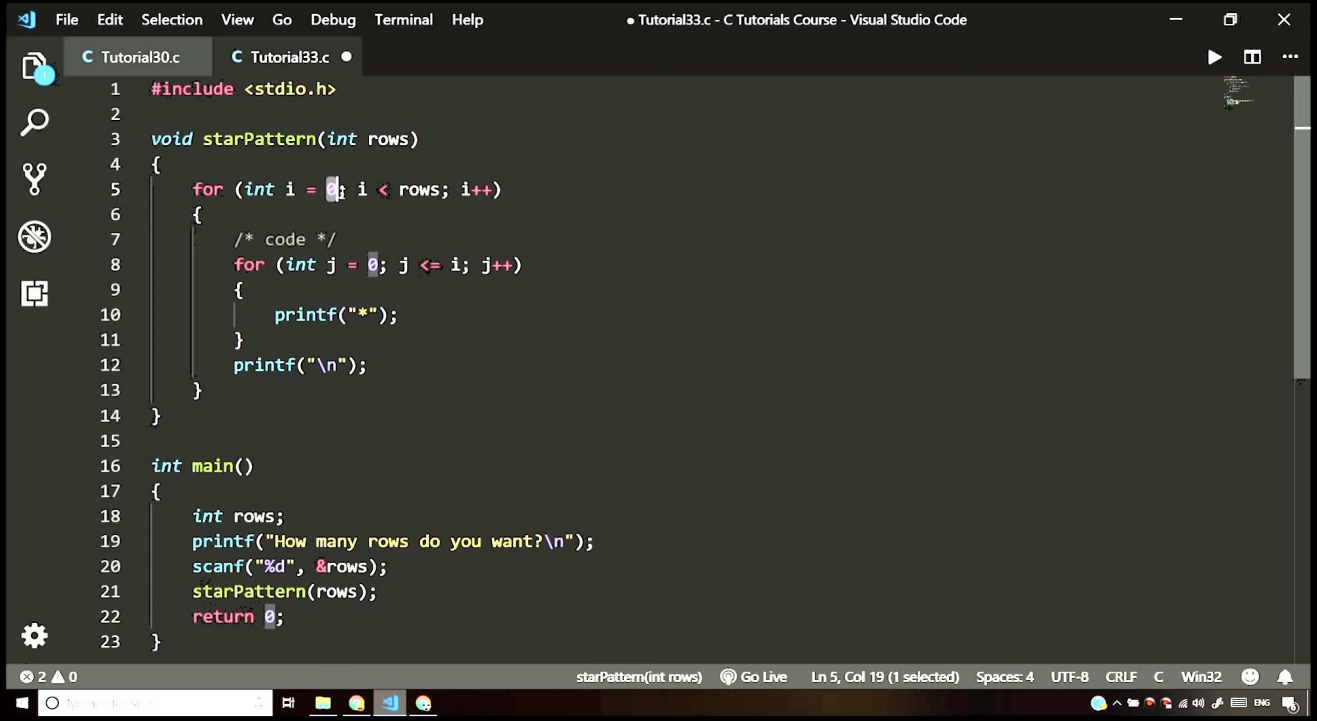
```
int avg(int* ptr)
{
  *(ptr) = 1
  *(ptr+1) → 2
  *(ptr+2) = 3
}
```

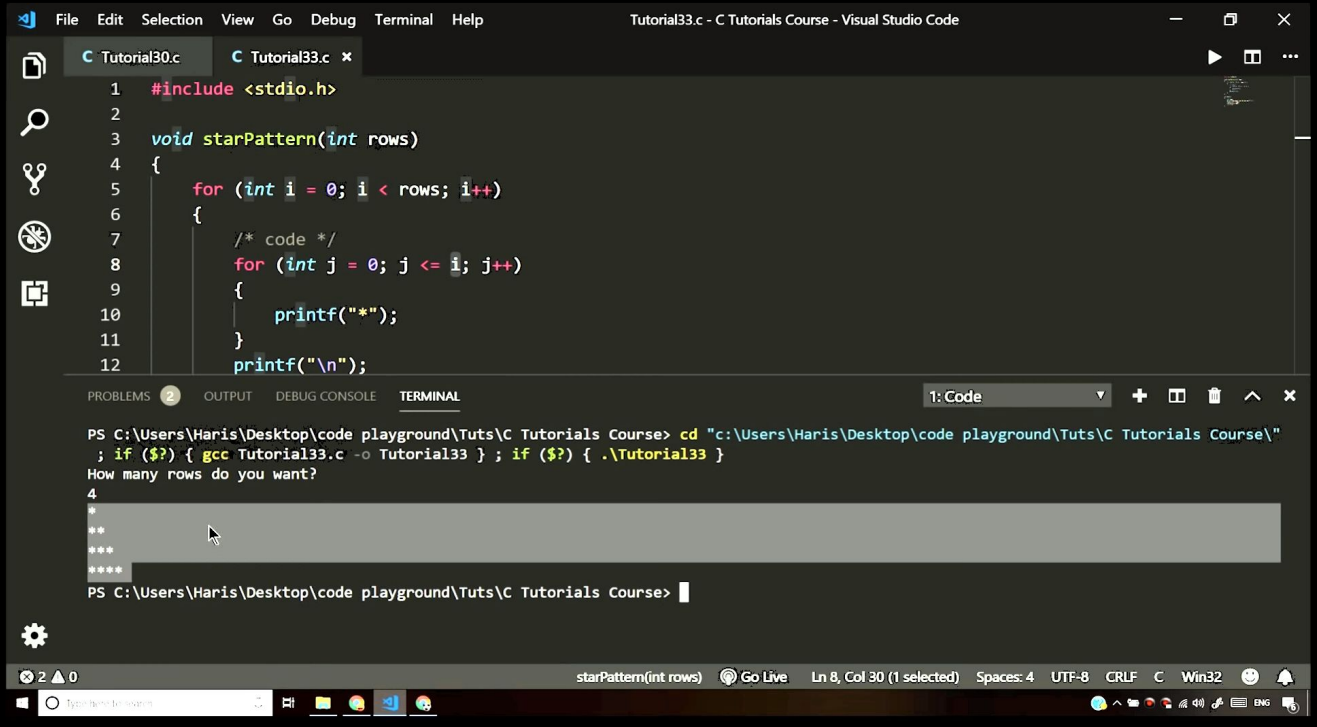
ARRAYS ARE PASSED TO FUNCTIONS THROUGH BASE ADDRESS

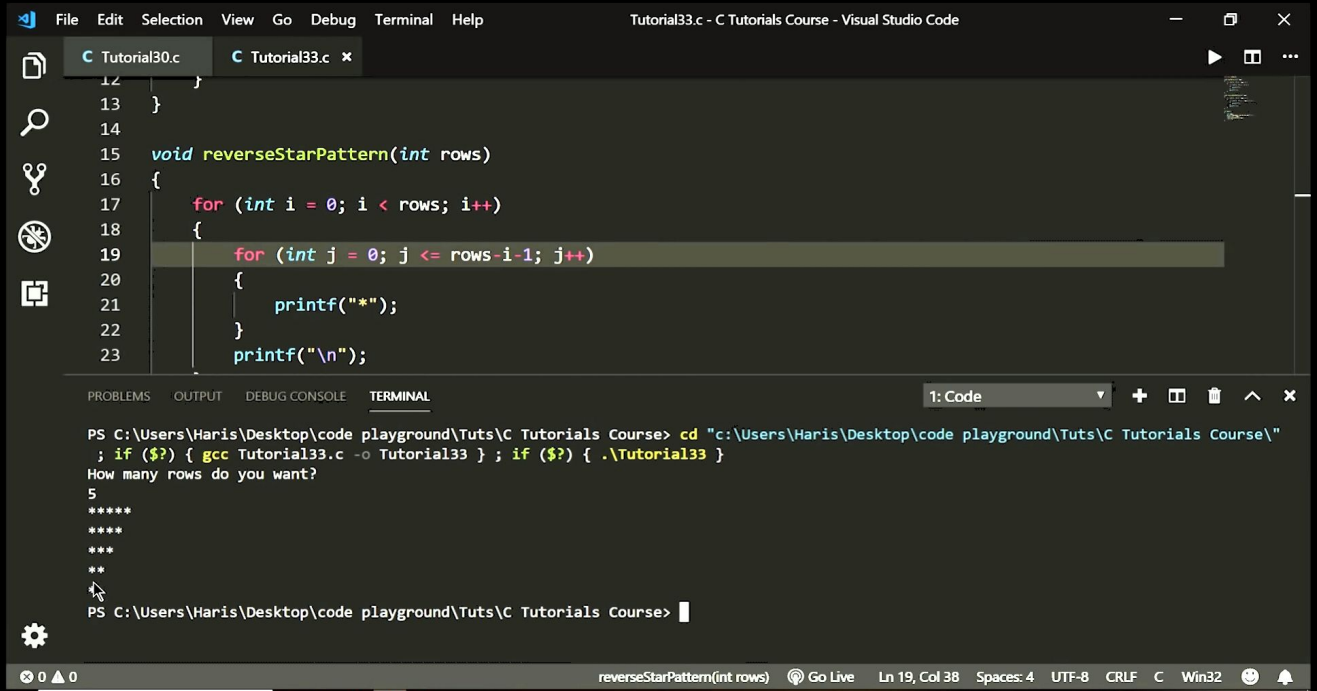


* arr = Value at arr (1000)

*(arr+1) = Value at arr+1 (1002)


File Edit Selection View Go Debug Terminal Help
● Tutorial33.c - C Tutorials Course - Visual Studio Code
Tutorial30.c
Tutorial33.c ●
#include <stdio.h>
void starPattern(int rows)
{
for (int i = 0; i < rows; i++)
{
/* code */
for (int j = 0; j <= i; j++)
{
printf("*");
}
printf("\n");
}
}
int main()
{
int rows;
printf("How many rows do you want?\n");
scanf("%d", &rows);
starPattern(rows);
return 0;
}
starPattern(int rows)
Go Live
Ln 5, Col 19 (1 selected)
Spaces: 4
UTF-8
CRLF
C
Win32
Subscribe


File Edit Selection View Go Debug Terminal Help
Tutorial33.c - C Tutorials Course - Visual Studio Code
Tutorial30.c
Tutorial33.c
#include <stdio.h>
void starPattern(int rows)
{
for (int i = 0; i < rows; i++)
{
/* code */
for (int j = 0; j <= i; j++)
{
printf("*");
}
printf("\n");
PROBLEMS 2
OUTPUT
DEBUG CONSOLE
TERMINAL
1: Code
PS C:\Users\Haris\Desktop\code playground\Tuts\C Tutorials Course> cd "c:\Users\Haris\Desktop\code playground\Tuts\C Tutorials Course\" ; if ($?) { gcc Tutorial33.c -o Tutorial33 } ; if ($?) { .\Tutorial33 }
How many rows do you want?
4
*
**
***
****
PS C:\Users\Haris\Desktop\code playground\Tuts\C Tutorials Course>
2 0
starPattern(int rows)
Go Live
Ln 8, Col 30 (1 selected)
Spaces: 4
UTF-8
CRLF
C
Win32
Subscribe

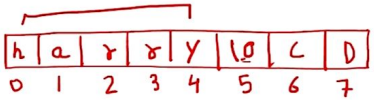

File Edit Selection View Go Debug Terminal Help
Tutorial33.c - C Tutorials Course - Visual Studio Code
Tutorial30.c
Tutorial33.c
}
}
void reverseStarPattern(int rows)
{
for (int i = 0; i < rows; i++)
{
for (int j = 0; j <= rows-i-1; j++)
{
printf("*");
}
printf("\n");
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: Code
PS C:\Users\Haris\Desktop\code playground\Tuts\C Tutorials Course> cd "c:\Users\Haris\Desktop\code playground\Tuts\C Tutorials Course\" ; if ($?) { gcc Tutorial33.c -o Tutorial33 } ; if ($?) { .\Tutorial33 }
How many rows do you want?
5
*****
****
***
**
*
PS C:\Users\Haris\Desktop\code playground\Tuts\C Tutorials Course>
0 0
reverseStarPattern(int rows)
Go Live
Ln 19, Col 38
Spaces: 4
UTF-8
CRLF
C
Win32

IS STRING A DATA TYPE IN C?

char name[27];

name = "Harry";
city = "Delhi";

- No
- We have char, int, float and other data types but no 'string' data type in C
- String is not a supported data type in C but it is a very useful concept used to model real world entities like name, city etc.
- We express strings using an array of characters terminated by a null character ('\0'). (len + 1)



"harryare"

X → null

WHAT ARE STRINGS IN C

'\0'

- String: array of characters terminated by NULL character
- Strings in c is created by creating an array of characters
- We need an extra character ('\0' or null character) to tell the compiler that the string ends here.

"harry" 6

CREATING STRINGS IN C

- We can create a character array in the following ways:
  - Char name[] = "harry";
  - Char name[] = {'h','a','r','r','y','\0'};

char name[] = {'c','o','d','e','\0'}

print remaining chars

Stop

TAKING STRING INPUT FROM THE USER

```
char str[52];
gets(str);
printf("%s", str);
puts(str); //aliter
```

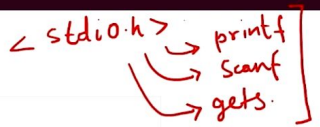
```c
// Online C compiler to run C program online
#include <stdio.h>

int main() {
    // char str[] = "Raihan";
    // char str[20] = "Raihan"
    // char str[] = {'R', a', 'i', 'h', 'a',
        'n', '\0'};
    // char str[7] = {'R', 'a', 'i', 'h', 'a',
        'n', '\0'};
    char str[20];
    scanf("%s", &str);
    printf("%s\n", str);

    gets(str);
    puts(str);

    return 0;
}
```

```
/tmp/OF7J4MtOxo.o
Raihan
Raihan

Radfgh
dash: 2: Radfgh: not found
```

WHAT ARE STRINGS IN C

- C language does not support strings as a data type.
- We express strings using an array of characters terminated by a null character ('\0').
- String: array of characters terminated by NULL character
- We can create a character array using the following ways:
  - Char name[] = "harry";
  - Char name[] = {'h', 'a', 'r', 'r', 'y', '\0'};

C LIBRARY: <string.h>



Function	Use
strcat()	This function is used to concatenate or combine two given strings
strlen()	This function is used to show length of a string
strrev()	This function is used to show reverse of a string
strcpy()	This function is used to copy one string into another
strcmp()	This function is used to compare two given strings

Reverse an Array in C

The task is to reverse the elements of the given array. We can use many approaches to do this, and it is a basic problem. It is one of the most frequently asked questions in interviews. This tutorial lists all the possible approaches to reversing an array in C.

1. Declaring another array
2. Iteration and Swapping
3. Using pointers

C Language Array Reversal Exercise 5: Solution: C Tutorial In Hindi #40

Today we are going to discuss the solution of problem given in Tutorial #36. I hope that you have solved the problem and have learned the usage of arrays completely. The problem and instructions are again displayed below:-

Instruction:

In this task you have to write a C program that will reverse an array of integers. For that purpose, create the function that will take array as an argument and reverse an array.
Your program should print the array before and after reversal.

For Example:

Before Reversal: 1, 2, 3, 4, 5, 6, 67
After Reversal: 67, 6, 5, 4, 3, 2, 1

Code as described/written in the video

Copy

```c
#include <stdio.h>
// target: 67,6,5,4,3,2,1

// 7
// 1,2,3,4,5,6,67
// 67,2,3,4,5,6,1
// 67,6,3,4,5,2,1
// 67,6,5,4,3,2,1


void arrayRev(int arr[])
{
    int temp;
    for (int i = 0; i < 7/2; i++)
    {
        //swap item arr[i] with arr[6-i]
        temp = arr[i];
        arr[i] = arr[6-i];
        arr[6-i] = temp;
    }
}

void arrayPrint(int arr[])
{
    for (int i = 0; i < 7; i++)
    {
        printf("The value of element %d is %d\n", i, arr[i]);
    }
}
int main()
{
    int arr[] = {1,2,3,4,5,6,67};
    printf("Before reversing the array\n");
    arrayPrint(arr);
    arrayRev(arr);
    printf("\nAfter reversing the array\n");
    arrayPrint(arr);
```

LOCAL VARIABLES (RECAP)

- Scope is a region of the program where a defined variable can exist and beyond which it cannot be accessed.
- Variables which are accessed inside a function or a block are called **local variables.**
- They can only be accessed by the function they are declared in!
- They are inaccessible to the functions outside the function they are declared in!

```
int main(){
    int a; int b;
a = myfunc(a);
}
```

```
int myfunc(int a){
    int a2 = 471;
}
```

GLOBAL VARIABLES (RECAP)

- These are the variables defined outside the main method.
- Global variables are accessible throughout the entire program from any function.
- If a local and global variable has the same name, the local variable will take preference.

```
int g1;
g1 = 7;
int main() {
    // some code
    return 0;
}
```

```
int func() {
    // int g1 = 3;
    printf("%d", g1);
}
```

STATIC VARIABLES IN C

static data_type name = value;
static int harry = 7;

- Static variables are variables which have a property of preserving their values even when they go out of scope.
- They preserve their value from the previous scope and are not initialized again.
- Static variable remains in memory throughout the span of the program.
- Static variables are initialized to 0 if not initialized explicitly.
- In C, static variables can only be initialized using constant literals.

// static int b = func1();
b = 5;

func1() {
  [static int a=5;
   a++;]
  return a;
}

main() {
a = func1()
printf("%d", a); // 6
a = func1()
printf a // 7

Mo | Tu | We | Th | Fr | Sa | Su

Date ....

- C programming was developed by Dennis Ritchie in 1972 at AT&T Bell Labs
- AT&T - American Telephone & Telegraph
- Extension used - main.c
- IDE used - VS code
- Compiler used - MINGW
- How to run your code
  - gcc main.c or gcc main.c -o raihon
  - .\a.exe or .\raihon.exe

Q- What really happens when a c program runs

main.c — Source File

↓

main.i — Preprocessor → Removal of Comments, Expansion of Macros, Expansion of the included files, Conditional Compilation

↓

main.s — Compiler → Preprocessed file is converted into assembly level instructions

↓

main.o — Assembler → Assembly level instructions is converted into machine level instruction. main.o is in binary 0's & 1's

↓

Linker → Function calls are linked with their function definition

WHAT & WHY DYNAMIC MEMORY ALLOCATION?

char name [39];

Shubham \0

harry \0

Shyam \0

- An statically allocated variable or array has a fixed size in memory.
- We have learnt to create big enough arrays to fit in our inputs but this doesn't seem like an optimal way to allocate memory.
- Memory is a very useful resource.
- Clearly we need a way to request memory on runtime.
- **Dynamic Memory Allocation** is a way in which the size of a data structure can be changed during the runtime.

STATIC VS DYNAMIC MEMORY ALLOCATION

int i;

Static Memory Allocation

- Allocation is done before the program's execution
- There is no memory reusability and the memory allocated cannot be freed.
- Less efficient

Dynamic Memory Allocation

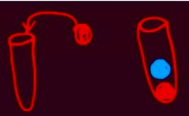
- Allocation is done during the program's execution
- There is memory reusability and the allocated memory can be freed when not required
- More efficient

MEMORY ALLOCATION IN C PROGRAMS

Quick →

- Memory assigned to a program in a typical architecture can be broken down into four segments:
  1. Code → Text segment
  2. Static/global variables → data segment (initialized storage) ✓ / bss segment (uninitialized segment) ✓
  3. Stack →
  4. Heap → dynamic memory allocation

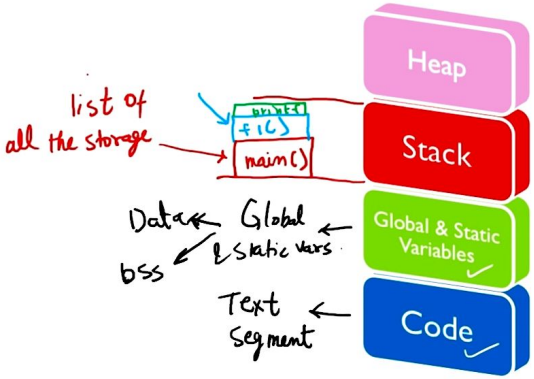
C PROGRAM: MEMORY LAYOUT



push red
push blue.
LIFO

```c
#include <stdio.h>

int b = 34; //global.
int ret()
{
      return 43*3;
}

int func1(int b1)
{
      static int myvar = 45;
      printf("The value of myvar is %d\n", myvar);
      myvar++;
      return b1 + myvar;
}
int main(){
      int b = 344;
      int val = func1(b);
      val = func1(b);
      val = func1(b);
      val = func1(b);
      val = func1(b);
      int *ptr = &val;
      return 0;
}
```

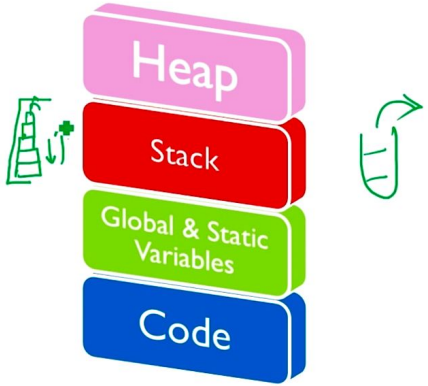
Initially some memory will be reserved for main() in the stack. This is also called as the stack frame of main()



C PROGRAM: STACK OVERFLOW

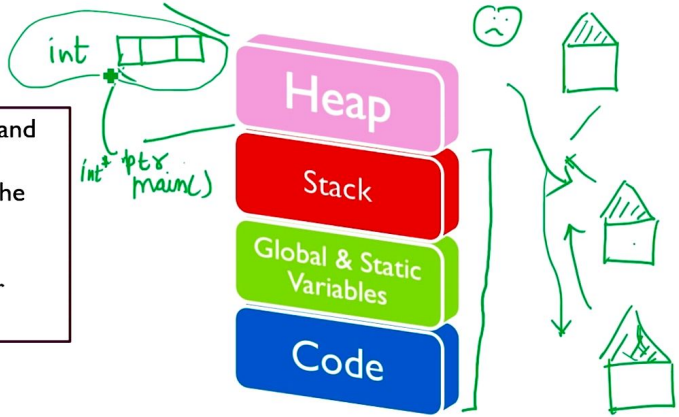
func1()
func1()

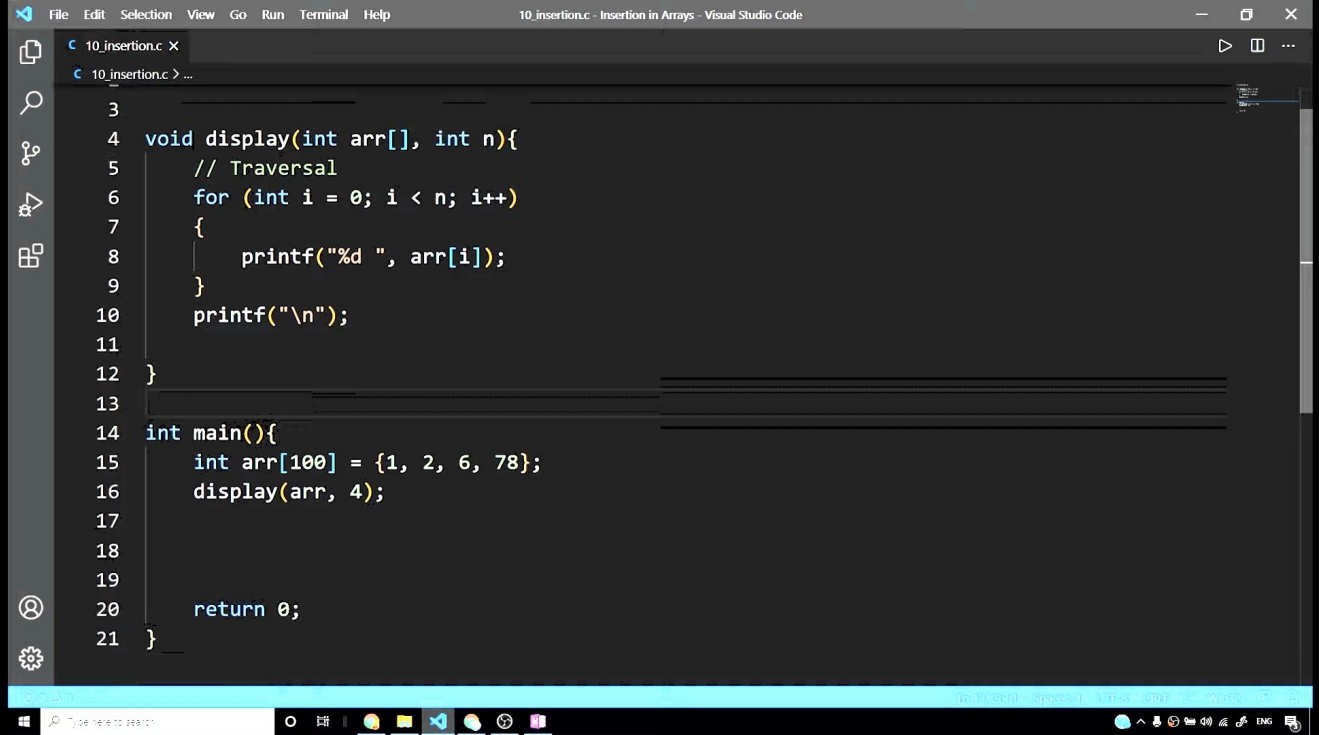
- Compiler allocates some space for the stack part of the memory
- When this space gets exhausted for some bad reason, the situation is called as stack overflow
- Typical example includes recursion with wrong/no base condition.



C PROGRAM: USE OF HEAP

- We can create a pointer in our main function and point to a memory block in the heap
- The address is stored by the local variable in the main function.
- The memory consumed will not get freed automatically in case we overwrite the pointer

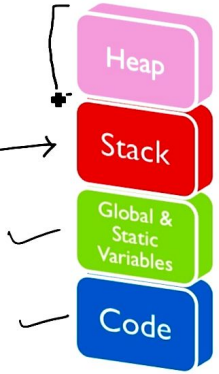



File Edit Selection View Go Run Terminal Help
10_insertion.c - Insertion in Arrays - Visual Studio Code
10_insertion.c
10_insertion.c > ...
3
4 void display(int arr[], int n){
5 // Traversal
6 for (int i = 0; i < n; i++)
7 {
8 printf("%d ", arr[i]);
9 }
10 printf("\n");
11
12 }
13
14 int main(){
15 int arr[100] = {1, 2, 6, 78};
16 display(arr, 4);
17
18
19
20 return 0;
21 }
Subscribe

DYNAMIC MEMORY ALLOCATION: RECAP

- An statically allocated variable or array has a fixed size in memory.
- **Dynamic Memory Allocation** is a way in which the size of a data structure can be changed during the runtime.
- Memory assigned to a program in a typical architecture can be broken down into four segments:
  1. Code
  2. Static/global variables
  3. Stack
  4. Heap

int arr[10];

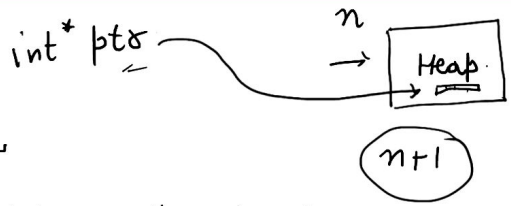


FUNCTIONS FOR DYNAMIC MEMORY ALLOCATION IN C

- In Dynamic memory allocation, the memory is allocated at runtime from the heap segment
- We have four functions that help us achieve this task:
  - malloc
  - calloc
  - realloc
  - free

→ Dynamic memory allocation

C MALLOC()

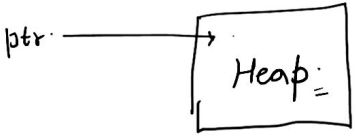


- malloc() stands for memory allocation
- It reserves a block of memory with the given amount of bytes.
- The return value is a void pointer to the allocated space
- Therefore the void pointer needs to be casted to the appropriate type as per the requirements
- However, if the space is insufficient, allocation of memory fails and it returns a NULL pointer.
- All the values at allocated memory are initialized to garbage values
  - Syntax:

```
ptr = (ptr-type*) malloc(size_in_bytes)
```

```
int* ptr;
ptr = (int*) malloc(3 * sizeof(int));
```

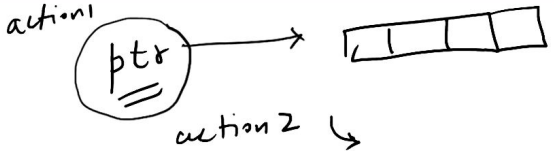
C CALLOC()



- calloc() stands for contiguous allocation
- It reserves n blocks of memory with the given amount of bytes.
- The return value is a void pointer to the allocated space
- Therefore the void pointer needs to be casted to the appropriate type as per the requirements
- However, if the space is insufficient, allocation of memory fails and it returns a NULL pointer.
- All the values at allocated memory are initialized to 0
  - Syntax:

```
ptr = (ptr-type*) calloc(n, size_in_bytes)
```

C REALLOC()



- realloc() stands for reallocation
- If the dynamically allocated memory is insufficient we can change the size of previously allocated memory using realloc() function
- Syntax:

```
ptr = (ptr-type*) realloc(ptr, new_size_in_bytes)
```

36 bytes

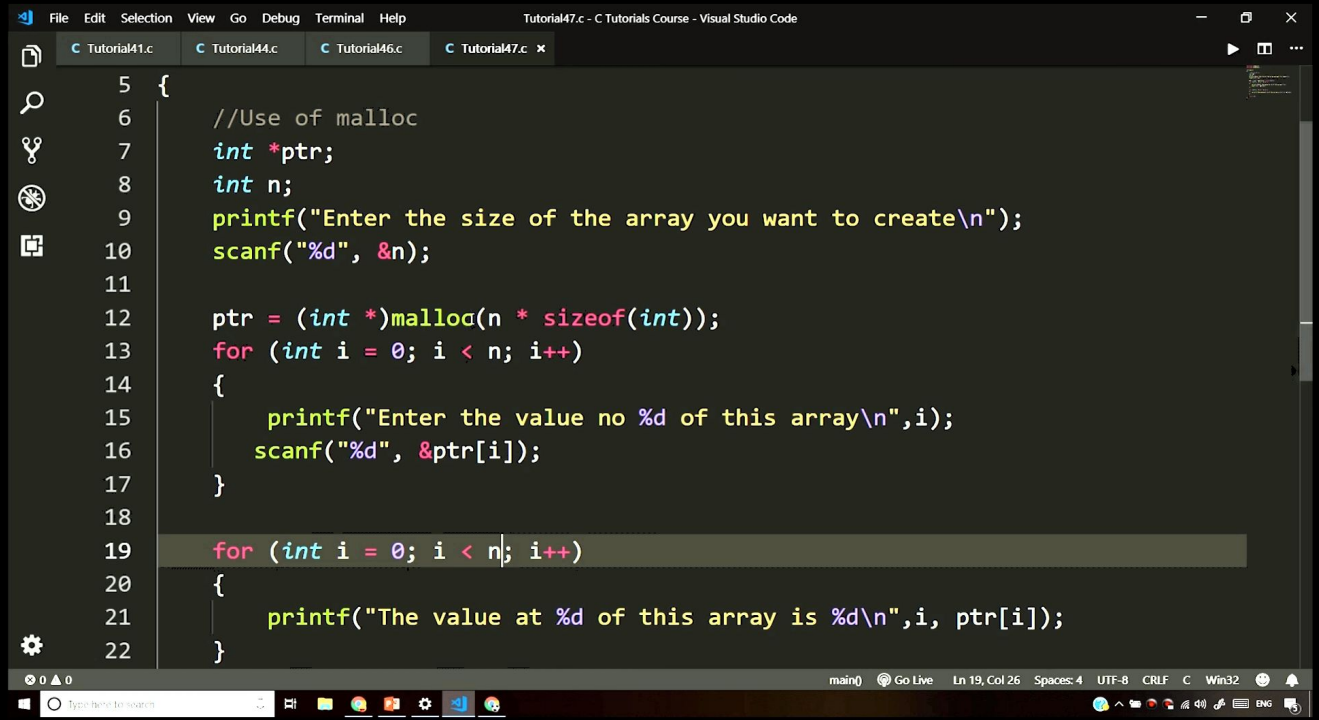
40 bytes

C FREE()

ptr = (int*) malloc(3 * sizeof(int))

- free() is used to free the allocated memory
- If the dynamically allocated memory is not required anymore, we can free it using free function.
- This will free the memory being used by the program in the heap
- Syntax:

```
free(ptr)
```


File Edit Selection View Go Debug Terminal Help
Tutorial47.c - C Tutorials Course - Visual Studio Code
Tutorial41.c
Tutorial44.c
Tutorial46.c
Tutorial47.c
{
//Use of malloc
int *ptr;
int n;
printf("Enter the size of the array you want to create\n");
scanf("%d", &n);
ptr = (int *)malloc(n * sizeof(int));
for (int i = 0; i < n; i++)
{
printf("Enter the value no %d of this array\n",i);
scanf("%d", &ptr[i]);
}
for (int i = 0; i < n; i++)
{
printf("The value at %d of this array is %d\n",i, ptr[i]);
}
main() Go Live Ln 19, Col 26 Spaces: 4 UTF-8 CRLF C Win32
Subscribe

Code as described/written in the video

```c
    //Use of calloc
    int *ptr;
    int n;
    printf("Enter the size of the array you want to create\n");
    scanf("%d", &n);

    ptr = (int *)calloc(n ,  sizeof(int));
    for (int i = 0; i < n; i++)
    {
        printf("Enter the value no %d of this array\n",i);
       scanf("%d", &ptr[i]);
    }

    for (int i = 0; i < n; i++)
    {
        printf("The value at %d of this array is %d\n",i, ptr[i]);
    }

    //Use of realloc
    printf("Enter the size of the new array you want to create\n");
    scanf("%d", &n);

    ptr = (int *)realloc(ptr ,  n*sizeof(int));
    for (int i = 0; i < n; i++)
    {
        printf("Enter the new value no %d of this array\n",i);
       scanf("%d", &ptr[i]);
    }

    for (int i = 0; i < n; i++)
    {
        printf("The new value at %d of this array is %d\n",i, ptr[i]);
    }

    free(ptr);


    return 0;
}
```