Migrating to ‘C’

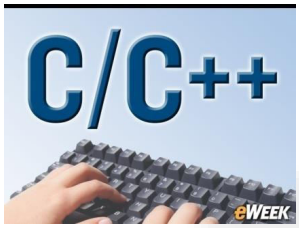
Why did We Learn Python?

- Easy to learn
- Language with simple rules
- Good for beginners
- Code is readable
- Less development time
- No memory management
- Great support for building web apps
- Dynamic language and no type checking

Limitations of Python

Python is not a good choice for:

- **Memory intensive and computation intensive tasks**
- **Embedded Systems where processor has limited capacity**
- **For graphic intensive 3D game that takes up a lot of CPU**
- **Applications that demand concurrency and parallelism**
- **Developing mobile apps**
- **Design restrictions**
- **Interpreted language and is slow compared to C/C++ or Java**



Why to learn more languages?

- **Similar to why a carpenter has more than just a hammer in his/her toolbox**
- **Every programming language has its positive and negative points**
- **One language cannot do everything**
- **That is why there are many languages; some are fantastic for some things**
  - **Eg: C/C++ is typically the benchmark for speed and memory usage, and some languages provide strengths elsewhere (Eg: Python is very easy to pick up)**

Transiting from Python to C/C++

- Will not be so hard
- There are quite a few syntax differences between the two languages
- Only way to learn a new programming language is by writing programs in it

– Dennis Ritchie

History of C

- Born at AT & T Bell Laboratory of USA in 1972
- Many of C's principles and ideas were derived from the earlier language B
- Ken Thompson was the developer of B Language
- C was written by Dennis Ritchie
  http://www.nytimes.com/2011/10/14/technology/dennis-ritchie-programming-trailblazer-dies-at-70.html?_r=0
- C language was created for a specific purpose i.e designing the UNIX operating system (which is currently base of many UNIX based OS)
- Quickly spread beyond Bell Labs in the late 70's because of its strong features

About Dennis Ritchie

- Born September 9, 1941
- Known for ALTRAN, B, BCPL, C, Multics, Unix
- Won Turing Award in 1983
- Developed C language which is widely used developing, operating systems, compiler, and embedded system development, Assemblers, Text editors, Print Spoolers, Network drivers databases etc and its influence is seen in most modern programming languages
- Died on October 12, 2011

Features of C language

- Portability - C Programs can run on any compiler with little or no modification
- Low level features: C provides low level features and is closely related to lower level a**ssembly Languages(**DOS operating systems, the Turbo **Pascal** compiler)
- Modular programming - software design technique that increases the extent to which software is composed of separate parts, called **modules**
- Has many successor languages which are designed to look like C, e.g., C++, C#, Objective-C, Java, JavaScript, PHP and Perl.

C is a structured programming language

- Divides the large problem in to smaller modules called functions or procedures
- Each function or module handles the particular task and the collection of all the functions is called a program, which solves the large problem
- Easier to modify and debug

Difference between Python and C

- **C programs – Compiled**
- **Python programs – Interpretered**

Compiler	Interpreter
Takes **entire** program as input and generate a output file with object code	Takes instruction by instruction as input and gives an output. But does not generate a file
Errors are displayed after **entire program** is checked	**Errors** are displayed for **every instruction** interpreted (if any)

Variable Declaration in C

- In C, it is mandatory to do variable declaration
- We say variable's **type**, whether it is an integer (**int**), floating-point number (**float**), character (**char**) etc
- Syntax is type of variable, white space, name of variable semicolon
- Eg: int number;

Intendation

- Block of code in C need not be intended as in Python
- In C, Curly braces are used for giving a block of code

  Eg: Block of code in ‘C’

```
{
-----
}
```

Problem

- Little Bob loves chocolate, and he goes to a store with Rs. *N* in his pocket. The price of each chocolate is Rs. *C*. The store offers a discount: for every *M* wrappers he gives to the store, he gets one chocolate for free. This offer is available only once. How many chocolates does Bob get to eat?

PAC For Chocolate Problem

Input	Processing	Output
Amount in hand, N Price of one chocolate, C Number of wrappers for a free chocolate, M	Number of Chocolates P = Quotient of N / C Free chocolate F = Quotient of P/M	Total number of chocolates got by Bob

Pseudocode

- READ N and C
- COMPUTE num_of_chocolates as N/C
- CALCULATE returning_wrapper as number of chocolates/m
- TRUNCATE decimal part of returning_wrapper
- COMPUTE Chocolates_recieved as num_of_chocolates + returning_wrapper
- PRINT Chocolates_recieved

Knowledge Required

- Following knowledge is required in C to write a code to solve the above problem
- Read input from user
- Data types in C
- Perform arithmetic calculations
- Write output

Python Program for Bob Problem

```
n = float(input("Enter amount in hand"))
c = float(input("Enter price of one chocolate"))
m = int(input("Enter num of wrapper for free chocolate"))
#compute number of chocolates bought
p = n//c
#compute number of free chocolates
f = p//m
print("Number of chocolates",int(p+f))
```

Layout of a C program

pre-processor directives – Preceded by a '#'
global declarations – Optional
int main() - standard start for all C programs
{
local variables to function main ; - all variables used in the function must be declared in the beginning
statements associated with function main ;
}
void f1()
{
local variables to function 1 ;
statements associated with function 1 ;
}

Components of a C program

A C program consists of the following parts:

- Comments
- Variables
- Preprocessor Commands
- Functions
- Statements & Expressions

Comments in C

Two types of comments

Single line and multi line comment

Single Line Comment is double forward slash ‘//’ and can be **Placed Anywhere**

Multiline Comments in C

- Multi line comment can be placed anywhere
- Multi line comment starts with /*
- Multi line comment ends with */
- Any symbols written between '/*' and '*/'

are ignored by Compiler

Data types in C

Basic Arithmetic types - further classified into: (a) integer types and (b) floating-point types

Enumerated types - arithmetic types that are used to define variables that can be assigned only certain discrete integer values throughout the program

Type void - indicates that no value is available

Derived types - They include (a) Pointer types, (b) Array types, (c) Structure types, (d) Union types and (e) Function types

Broad Classification of Data Types

- Numerical data types are broadly classified into
  - Signed
  - Unsigned
- Signed can store zero, positive and negative values
- Unsigned can store only zero and positive values
- Some applications use unsigned data types only Eg: age

Integer Types

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

Floating Point Types

Type	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

Keywords

- 32 keywords available in C

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

Compiler vendors (like Microsoft, Borland, etc.) provide their own keywords

Discuss Valid and Invalid variable names

- BASICSALARY
- _basic
- basic-hra Invalid
- #MEAN Invalid
- group. Invalid
- 422 Invalid
- population in 2006 Invalid
- FLOAT
- hELLO

I/O in C

- Basic operation in any language
- Input is got through a function scanf which is equivalent to input or raw_input in Python
- Scanf has two or more arguments
- First format string, followed by address of variables that are going to hold values entered by user

I/O in C

- printf contains one or more arguments
- first argument is the format string

printf and scanf format codes

code	type	format
d	int	decimal (base ten) number
o	int	octal number (no leading '0' supplied in printf)
x or X	int	hexadecimal number (no leading '0x' supplied in printf; accepted if present in scanf) (for printf, 'X' makes it use upper case for the digits ABCDEF)
ld	long	decimal number ('l' can also be applied to any of the above to change the type from 'int' to 'long')

printf and scanf format codes

code	type	format
u	Unsigned int	decimal number
lu	unsigned long	decimal number
c	char	single character
s	char pointer	string
f	float	number with six digits of precision
lf	double	number with six digits of precision

Address of a variable

- Address of a variable can be obtained by putting a ‘&’ before the variable name

Example 1

```
#include<stdio.h>
void main()
{
int a = 27;
int b = 25;
int c = a -b;
printf("%d",c);
}
```

Output 1

2

Example 2

```
#include<stdio.h>
void main()
{
int a,b,c;
scanf("%d%d",&a,&b);
c=a+b;
printf("%d",c);
}
```

Output 2

Input:
2
3
Output:
5

Example 3

```
#include<stdio.h>
void main()
{
char a = 27;
char b = 25;
char c = a -b;
printf("%c",c);

}
```

Output 3

A special character

Arithmetic Operators in C

Operator	Description
+	Adds two operands
–	Subtracts second operand from the first.
*	Multiplies both operands
/	Divides numerator by denominator
%	Modulus Operator and remainder of after an integer division.
++	Increment operator increases the integer value by one
--	Decrement operator decreases the integer value by one

Logical Operators		
Operator	Description	Example
&&	AND	x=6 y=3 x<10 && y>1 Return True
||	OR	x=6 y=3 x==5 || y==5 Return False
!	NOT	x=6 y=3 !(x==y) Return True

Operation	Meaning
x & y	Bitwise AND
x | y	Bitwise OR
x ^ y	Bitwise XOR
~x	Invert all bits of x
x >> y	Shift all bits of x y positions to the right
x << y	Shift all bits of x y positions to the left

Precedence of Operators in C

++, -- Post increment Operators

++, -- Pre increment Operators

Operators	Description
* / %	multiplication, division, modular division
+ -	addition, subtraction
=	assignment

Parenthesis can be used to override default precedence

Example 4

```
#include<stdio.h>
main()
{
int a, b,c;
a = 4;
b = 2;
c = -a+--b;
printf ( "c = %d", c) ;
}
```

Output 4

```
c = -3
```

Example 5

```
#include<stdio.h>
main()
{
int a, b,c;
a = 4;
b = 2;

c = -a+ b--;

printf ( "c = %d", c) ;
printf ( "b = %d", b) ;
}
```

Output 5

```
c = -2
b = 1
```

Example 6

```
#include<stdio.h>
main()
{
int a, b,c;
a = 4;
c = a++;
printf ( "c = %d", c) ;
printf ( "a = %d", a) ;
}
```

Output 6

c = 4 a = 5

Example 7

```
#include<stdio.h>
main()
{
int a, b,c;
a = 4;
c =++a;
printf ( "c = %d", c) ;
printf ( "a = %d", a) ;
}
```

Output 7

c = 5 a = 5

Example 8

```
#include<stdio.h>
main()
{
int a, b,c;
a = 4;

c = a++ + ++a;

printf ( "c = %d", c) ;

printf ( "a = %d", a) ;

}
```

Output 8

c = 10 a = 6

Associativity of Operators in C

When precedence of two operators are same then associativity of operator is considered for evaluation

OPERATOR	DESCRIPTION	ASSOCIATIVITY
( )	Parentheses (function call) (see Note 1)	left-to-right
[ ]	Brackets (array subscript)	
.	Member selection via object name	
->	Member selection via pointer	
++ —	Postfix increment/decrement (see Note 2)	
++ —	Prefix increment/decrement	right-to-left
+ –	Unary plus/minus	
! ~	Logical negation/bitwise complement	
(*type*)	Cast (convert value to temporary value of *type*)	
*	Dereference	
&	Address (of operand)	
sizeof	Determine size in bytes on this implementation	
* / %	Multiplication/division/modulus	left-to-right
+ –	Addition/subtraction	left-to-right
<< >>	Bitwise shift left, Bitwise shift right	left-to-right
< <=	Relational less than/less than or equal to	left-to-right
> >=	Relational greater than/greater than or equal to	
== !=	Relational is equal to/is not equal to	left-to-right
&	Bitwise AND	left-to-right

|	Bitwise inclusive OR	left-to-right
&&	Logical AND	left-to-right
||	Logical OR	left-to-right
? :	Ternary conditional	right-to-left
=	Assignment	right-to-left
+= -=	Addition/subtraction assignment	
*= /=	Multiplication/division assignment	
%= &=	Modulus/bitwise AND assignment	
^= |=	Bitwise exclusive/inclusive OR assignment	
<<= >>=	Bitwise shift left/right assignment	
,	Comma (separate expressions)	left-to-right

Automatic Type Conversion in C

Convert a variable from one data type to another data type

When the type conversion is performed automatically by the compiler without programmers intervention, such type of conversion is known as **implicit type conversion** or **type promotion**.

The compiler converts all operands into the data type of the largest operand.

Rules for Implicit Type Conversion in C

• Sequence of rules that are applied while evaluating expressions are given below:
• All short and char are automatically converted to int, then,
• If either of the operand is of type long double, then others will be converted to long double and result will be long double.
• Else, if either of the operand is double, then others are converted to double.
• Else, if either of the operand is float, then others are converted to float.

Example 9

```
#include<stdio.h>
void main()
{
char a = 65;
char b = 100;
char c = a%b;
printf("%c",c);

}
```

Output 9

A

Example 10

```
#include<stdio.h>
void main()
{
int a = 165;
int b = 100;
int c = (float)a/b;
printf("%d",c);

}
```

Output 10

1

Example 11

```
#include<stdio.h>
void main()
{
int a = 165;
int b = 100;
float c = a/b;
printf("%f",c);

}
```

Output 11

1.000000

Example 12

```
#include<stdio.h>
void main()
{
float a = 165;
float b = 100;
float c = a/b;
printf("%f",c);

}
```

Output 12

1.650000

Example 13

```
#include<stdio.h>
void main()
{
int a = 165;
float b = 100;
float c = a/b;
printf("%f",c);

}
```

Output 13

1.650000

Explicit Type Conversion

Type conversion performed by the programmer is known as explicit type conversion

Explicit type conversion is also known as **type casting**.

Type casting in c is done in the following form:

(data_type)expression;

where, *data_type* is any valid c data type,

and *expression* may be constant, variable or an expression

For example, x=(int)a+b*d;

Explicit Type Conversion

The following rules have to be followed while converting the expression from one type to another to avoid the loss of information:

- All integer types to be converted to float.
- All float types to be converted to double.
- All character types to be converted to integer.

Example 14

```
#include<stdio.h>
void main()
{
int a = 165;
int b = 100;
float c = (float)(a/b);
printf("%f",c);

}
```

Output 14

1.000000

Example 15

```
#include<stdio.h>
void main()
{
int a = 165;
int b = 100;
float c = (float) a/b;
printf("%f",c);

}
```

Output 15

1.650000

Declaration and Definition of Variables in C

- Declaration – Information to compiler about the variable
- Definition – Memory is allocated for the variable according to type

```c
#include <stdio.h>
void main()
{
    float n,c;
    int p,m,f,tot;
    printf("Enter amount in hand, price of chocolate and number of free wrappers");
    scanf("%f%f%d",&n,&c,&m);
    //compute number of chocolates bought
    p = (int)(n/c);
    //free chocolates
    f = (int)(p/m);
    tot = p+f;
    printf("Number of chocolates bought %d\n",tot);
}
```

```
janaki@janaki-HP-248-G1:~/CSE1001 Fall16$ gcc bob.c
janaki@janaki-HP-248-G1:~/CSE1001 Fall16$ ./a.out
Enter amount in hand, price of chocolate and number of free wrappers
10
3
2
Number of chocolates bought 4
janaki@janaki-HP-248-G1:~/CSE1001 Fall16$
```

Problem

ABC company Ltd. is interested to computerize the pay calculation of their employee in the form of Basic Pay, Dearness Allowance (DA) and House Rent Allowance (HRA). DA and HRA are calculated as certain % of Basic pay(For example, DA is 80% of Basic Pay, and HRA is 30% of Basic pay). They have the deduction in the salary as PF which is 12% of Basic pay. Propose a computerized solution for the above said problem.

Input : Basic Pay

Process : Calculate Salary

(Basic Pay + (Basic Pay * 0.8) + (Basic Pay * 0.3 - (Basic Pay * 0.12)

----------allowances ---------- --- deductions----

Output : Salary

Problem

- Find the average runs scored by a batsman in 4 matches
- Area of a circle

Exercise

An university is setting up a new lab at their premises. Design an algorithm and write Python code to determine the approximate cost to be spent for setting up the lab. Cost for setting the lab is sum of cost of computers, cost of furnitures and labour cost. Use the following formulae for solving the problem:

Cost of computer = cost of one computer * number of computers

Cost of furniture = Number of tables * cost of one table + number of chairs * cost of one chair

Labour cost = number of hours worked * wages per hour

Selection Structures

Syntax of if Statement Two Alternatives

Form 1:

if (condition)

statement $_T$;

Eg:

```
if (x > 0.0)
pos_prod = pos_prod * x;
```

If condition evaluates to true (a nonzero value), then statement $_T$ is executed; otherwise, statement $_T$ is skipped.

Syntax of if Statement Two Alternatives

Form 2:

if (condition)

statement $_T$;

else

statement $_F$;

Example

```
if (x >= 0.0)
printf("positive\n");
else
printf("negative\n");
```

If condition evaluates to true (a nonzero value), then $statement_T$ is executed and $statement_F$ is skipped; otherwise, $statement_T$ is skipped and $statement_F$ is executed.

Example 1

- ```
  #include<stdio.h>
  void main()
  {
  int i=1;
  if(i<1)
  printf("Yes");
  else
  printf(“No”);
  }
  ```
  ```

Output 1

- No

Example 2

- ```
  #include<stdio.h>
  void main()
  {
  int i=2;
  if(i)
  printf("Yes");
  else
  printf(“No”);
  }
  ```
  ```

Output 2

- Yes

Example 3

- ```
  #include<stdio.h>
  void main()
  {
  char i =‘a’;
  if(i)
  printf("Yes");
  else
  printf(“No”);
  }
  ```
  ```

Output 3

- Yes

Example 4

- ```
  #include<stdio.h>
  void main()
  {
  int a = 2 , b=5;
  if ((a==0)&&(b==1))
  printf(“Hi”);
  printf(“a is %d and b is %d”, a, b);
  }
  ```
  ```

Output 4

- a is 2 and b is 5

Example 5

- ```
  #include<stdio.h>
  void main()
  {
  int a = 2 , b=5;
  if ((a==2)&&(b==1))
  printf(“Hi”);
  printf(“a is %d and b is %d”, a, b);
  }
  ```
  ```

Output 5

- a is 2 and b is 5

Conditional Operator (Ternary Operator)

Syntax:

expression 1 ? expression 2 : expression 3

“if expression 1 is true (that is, if its value is non-zero), then the value returned will be expression 2, otherwise the value returned will be expression 3”.

Conditional Operator (Ternary Operator)

Example:

```
char   a ;
int   y ;
scanf ( "%c", &a ) ;
y = ( a >= 65 && a <= 90 ? 1 : 0 ) ;
```

Here 1 would be assigned to y if a >=65 && a <=90 evaluates to true, otherwise 0 would be assigned

Conditional Operator (Ternary Operator)

- Not necessary that conditional operators should be used only in arithmetic statements

Eg1:

```
int   i ;
scanf ( "%d", &i ) ;
( i == 1 ? printf ( "Amit" ) : printf ( "All and sundry" ) )
```

Eg2:

```
char   a = 'z' ;
printf ( "%c" , ( a >= 'a' ? a : '!' ) ) ;
```

Nested

- Conditional operators can be nested :

```
int   big, a, b, c ;
big = ( a > b ? ( a > c ? 3: 4 ) : ( b > c ? 6: 8 ) ) ;
```

Nested if Statements

if (condition$_1$)

statement$_1$

else if (condition$_2$)

statement$_2$

. . .

else if (condition$_n$)

statement$_n$

else

statement$_e$

Example

/* increment num_pos, num_neg, or num_zero depending on x */

if (x > 0)

num_pos = num_pos + 1;

else if (x < 0)

num_neg = num_neg + 1;

else /* x equals 0 */

num_zero = num_zero + 1;

```c
#include<stdio.h>
void main()
{
//Declare a variable to store ph value
float ph;
//Read the value of ph
scanf("%f",&ph);
//if ph is from 0 to 2 then print very acidic
if ((ph>=0)&&(ph<=2))
printf("Very acidic");
//if ph is greater than 2 and less than 7 then print acidic
else if((ph>2) &&(ph<7))
printf("Acidic");
//if ph is equal to 7 then print neutral
else if (ph==7)
printf("Neutral");
//if ph is greater than 7 and less than or equal to 12 then print alkaline
else if(ph>7&&ph<=12)
printf("Alkaline");
//if ph is greater than 12 then print Very alkaline
else
printf("Very alkaline");
}
```

#define statements

- These are preprocessor directives that are used define constants
- When a value is defined using #define, if there is a change in value it is easier to make modification

Syntax:

#define PI 3.14

```
#include<stdio.h>
#define l_Limit_VA 0
#define u_Limit_VA 2
#define l_Limit_A 2
#define u_Limit_A 7
#define l_Limit_Al 7
#define u_Limit_Al 12
void main()
{
//Declare a variable to store ph value
float ph;
//Read the value of ph
scanf("%f",&ph);
//if ph is from 0 to 2 then print very acidic
if ((ph>=l_Limit_VA)&&(ph<=u_Limit_VA))
printf("Very acidic");
//if ph is greater than 2 and less than 7 then print acidic
else if((ph>l_Limit_A) &&(ph<u_Limit_A))
printf("Acidic");
//if ph is equal to 7 then print neutral
else if (ph==7)
printf("Neutral");
//if ph is greater than 7 and less than or equal to 12 then print alkaline
else if(ph>l_Limit_Al&&ph<=u_Limit_Al)
printf("Alkaline");
//if ph is greater than 12 then print Very alkaline
else
printf("Very alkaline");
}
```

Switch Statement

- Useful when the selection is based on the value of a single variable or of a simple expression (called the controlling expression)
- Value of this expression may be of type int or char , but not of type double

Syntax of Switch Statement

switch (controlling expression)
{
label set_1
$statements_1$
break;
label set_n
$statements_n$
break;
default:
$statements_d$
}

Syntax of Switch Statement

- When a match between the value of the controlling expression and a case label value is found, the statements following the case label are executed until a break statement is encountered.
- Then the rest of the switch statement is skipped.
- If no case label value matches the controlling expression, the entire switch statement body is skipped unless it contains a default label.

```
#include <stdio.h>
int main()
{
int language = 10;
switch (language)
{
case 1:
printf("C#\n");
break;
case 2:
printf("C\n");
break;
case 3:
printf("C++\n");
break;
default:
printf("Other programming language\n");
}
}
```

Practice Problem

The table below shows the normal boiling points of several substances. Write a program that prompts the user for the observed boiling point of a substance in °C and identifies the substance if the observed boiling point is within 5% of the expected boiling point. If the data input is more than 5% higher or lower than any of the boiling points in the table, the program should output the message Substance unknown.

Substance	Normal boiling point (°C)
Water	100
Mercury	357
Copper	1187
Silver	2193
Gold	2660

C Loops

while loop in C

Syntax

- The syntax of a while loop in C programming language is:

```
while(condition)
{
    statement(s);
 }
```

Example

```c
#include <stdio.h>

int main ()
{
   /* local variable definition */
   int a = 10;

   /* while loop execution */
   while( a < 20 )
   {
      printf("value of a: %d\n", a);
      a++;
   }

   return 0;
}
```


for loop in C

- Syntax

The syntax of a for loop in C programming language is:

```
for ( init; condition; increment )
{
    statement(s);
}
```

Example

```c
#include <stdio.h>

int main ()
{
   /* for loop execution */
   for( int a = 10; a < 20; a = a + 1 )
   {
      printf("value of a: %d\n", a);
   }

   return 0;
}
```


do...while loop in C

- Unlike for and while loops, which test the loop condition at the top of the loop, the do...while loop in C programming language checks its condition at the bottom of the loop.
- A do...while loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

Syntax

- The syntax of a do...while loop in C programming language is:

```
do
{
   statement(s);
 } while( condition );
```

Example

```c
#include <stdio.h>

int main ()
{
   /* local variable definition */
   int a = 10;

   /* do loop execution */
   do
   {
       printf("value of a: %d\n", a);
       a = a + 1;
   }while( a < 20 );

   return 0;
}
```


nested loops in C

```c
for ( init; condition; increment )
{
   for ( init; condition; increment )
   {
      statement(s);
   }
   statement(s);
}
```

```c
while(condition)
{
   while(condition)
   {
      statement(s);
   }
   statement(s);
}
```

break statement in C

```c
#include <stdio.h>

int main ()
{
   /* local variable definition */
   int a = 10;

   /* while loop execution */
   while( a < 20 )
   {
      printf("value of a: %d\n", a);
      a++;
      if( a > 15)
      {
         /* terminate the loop using break statement */
         break;
      }
   }

   return 0;
}
```

result:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
```


continue statement in C

- The continue statement in C programming language works somewhat like the break statement. Instead of forcing termination, however, continue forces the next iteration of the loop to take place, skipping any code in between.

```c
#include <stdio.h>

int main ()
{
   /* local variable definition */
   int a = 10;

   /* do loop execution */
   do
   {
      if( a == 15)
      {
         /* skip the iteration */
         a = a + 1;
         continue;
      }
      printf("value of a: %d\n", a);
      a++;

   }while( a < 20 );

   return 0;
}
```

continue statement in C

Output:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```


goto statement in C

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

Syntax

```
goto label;
..
.
label: statement;
```

Example- goto statement

```c
#include <stdio.h>

int main ()
{
   /* local variable definition */
   int a = 10;

   /* do loop execution */
   LOOP:do
   {
      if( a == 15)
      {
         /* skip the iteration */
         a = a + 1;
         goto LOOP;
      }
      printf("value of a: %d\n", a);
      a++;

   }while( a < 20 );

   return 0;
}
```


Example-goto statement

Output:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

The Infinite Loop

- A loop becomes infinite loop if a condition never becomes false.

```
#include <stdio.h>

int main ()
{

   for( ; ; )
   {
      printf("This loop will run forever.\n");
   }

   return 0;
}
```

When the conditional expression is absent, it is assumed to be true. You may have an initialization and increment expression, but C programmers more commonly use the **for(;;)** construct to signify an infinite loop.

NOTE: You can terminate an infinite loop by pressing **Ctrl + C** keys.

Array and Strings

Accident Problem

Each year the Department of Traffic Accidents receives accident count reports from a number of cities and towns across the country. Given details of 'n' days, develop an algorithm and write a program to determine the average number of accidents and for each day, print the difference between the number of accidents on that day and average. For example, if the number of accidents is 5 and the values are 10, 12, 15, 13, 5 then average is 11 and the difference of values are 1, 1, 4, 2, 6

Accident Problem

Input	Output	Logic Involved
Value of 'n', the number of accidents	Average and 'n' values that is the difference between average and value	Find average and difference

Algorithm

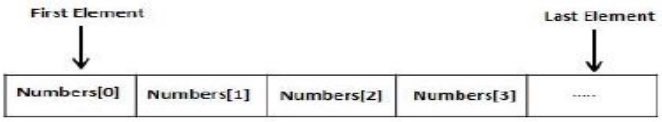
1. Read the value of ‘n’
2. Read the number of accidents happened in ‘n’ days
3. Find average
4. For each value print the difference between average and the value

New Stuff..

- We can find the difference between average and the number of accidents on a particular day only after reading all numbers from the user
- So data has to be stored
- Same type of data is to be stored
- Number of items not known prior
- Best choice would be using arrays in C
- Array - Can store a fixed-size sequential collection of elements of the same type

Arrays in C

- Consist of contiguous memory locations
- lowest address corresponds to the first element
- highest address to the last element
- Array indices start with zero
- The elements have indices from 0 to ‘n-1’
- Similar to list in Python but homogenous



Declaration in C

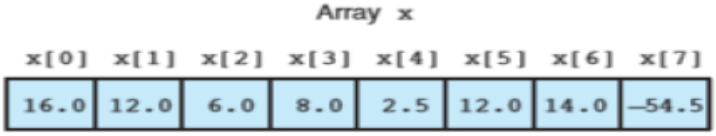
- type arrayName [arraySize];
- double balance[10];

Initializing Arrays

double balance[] = {1000.0, 2.0, 3.4, 7.0, 50.0};

(or)

double balance[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};



Cannot assign one array to other

```
int ia[] = {0, 1, 2}; // ok: array of ints
int ia2[] = ia; // error: cannot initialize one array
with another
int main()
{
const unsigned array_size = 3;
int ia3[array_size];
// ok: but elements are uninitialized!
ia3 = ia; // error: cannot assign one array to
another
return 0;
}
```

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
int acc_Details[20],total=0;
int num_Of_Days,counter;
float mean;
scanf("%d",&num_Of_Days);
for (counter=0;counter<num_Of_Days;counter++)
{
scanf("%d",&acc_Details[counter]);
total+=acc_Details[counter];
}
mean = total/(float)num_Of_Days;
printf("%.2f\n",mean);

for (counter=0;counter<num_Of_Days;counter++)
{
printf("%.2f\n",fabsf(mean-acc_Details[counter]));
}
}
```

Compare List and Array

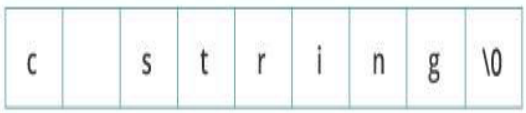
List	Arrays
Can have mixed type of elements	Can have only one type of element
Number of elements in list need not be specified L = []	Size has to be specified during declaration int a[10];
Elements are accessed by subscript operator L[0], L[1]...	Same way a[0], a[1], a[2],...
Size is dynamic, increases when elements are added and decreases when removed	Size is static
Have predefined functions such as len, count, index etc	No such functions

String

In C programming, a string is a sequence of characters terminated with a null character \0. For

```
char c[10] = "c string";
```

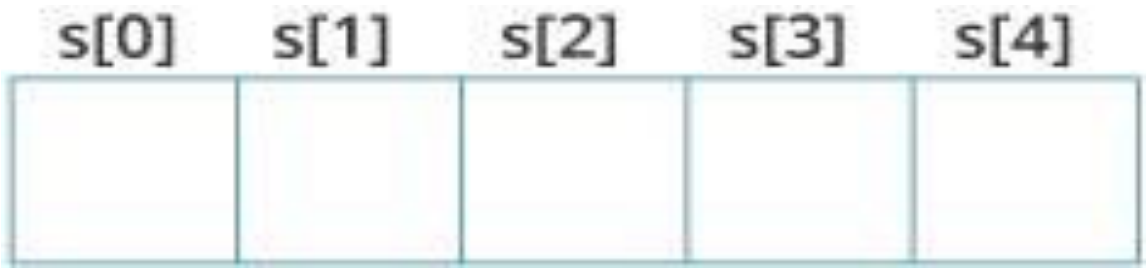
When the compiler encounters a sequence of characters enclosed in the double quotation marks, it appends a null character \0 at the end by default.



String Declaration

```
char s[5];
```

String with 5 characters



Assigning Values to Strings

```
char c[100];
c = "C programming"; // Error! array type is not assignable.
```

Copy the string

```
#include<stdio.h>
#include<string.h>
int main()
{
   char name[30]="Brijendra", name1[30]="Sanjay";
   strcpy(name,name1);
   printf("%s",name);
   return 0;
}
```

Example 1: scanf() to read a string

```
#include <stdio.h>
int main()
{
      char name[20];
      printf("Enter name: ");
      scanf("%s", name);
      printf("Your name is %s.", name);
      return 0;
}
```

Example 2: gets() and puts()

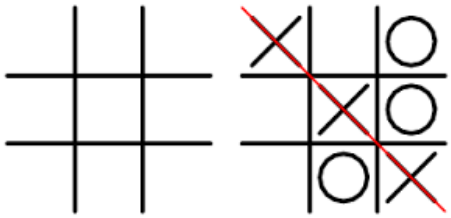
You can use the fgets() function to read a line of string. And, you can use puts() to display the string.

```
#include <stdio.h>
int main()
{
   char name[30];
   printf("Enter name: ");
   gets(name);  // read string
   printf("Name: ");
   puts(name);    // display string
   return 0;
}
```

Multi-D

Tic Tac Toe Problem

Tic-tac-toe is a paper-and-pencil game for two players, *X* and *O*, who take turns marking the spaces in a 3×3 grid. Player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row wins the game.



Tic Tac Toe Problem Contd...

Given the board configuration of the tic tac toe game, determine if the board is in either of the following states: empty, player1 wins, player2 wins, draw or intermediate. The board is said to be in initial state if all the cells contain '-1', player1 uses '1' as his coin and player2 uses '2' as his coin. The game is draw when the board is full and no one has won the game. The game is in intermediate state when no one has won and board is not full

Tic Tac Toe problem

Input	Output	Logic Involved
Current board configuration	State of the board as win, draw, initial or intermediate	Find average and difference

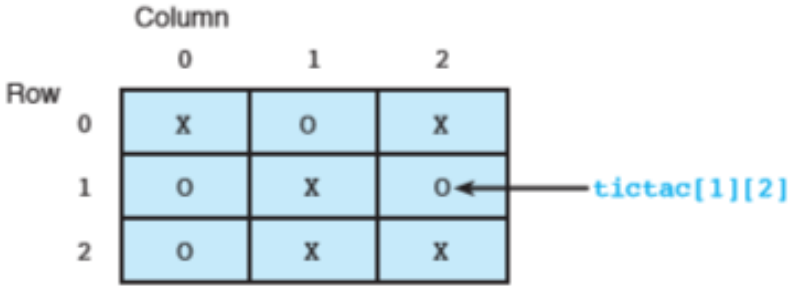
Algorithm

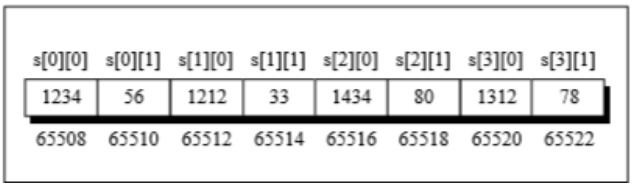
- Represent the board in memory
- Get the elements in first row, second row and so on
- Process the elements
- If all are -1 then print ‘empty’
- If ‘1’ is placed row wise, column wise or diagonally then print ‘Player 1’ wins
- If ‘2’ is placed row wise, column wise or diagonally then print ‘Player 2’ wins
- If all cells are full and no one has won the game then print ‘Draw’
- Otherwise print intermediate

New Stuffs...

- Represent the board in memory using a 2 d array
- Traverse the board using nested loop
- Memory is only a 1 d structure
- But high level languages supports arrays of multiple dimensions
- A kind of ordering is done internally to support multi dimensional arrays
- Either row major or column major ordering is done
- ‘C’ does row major ordering for 2 D arrays

Representation of Tic Tac Toe Board





Row Major Ordering of 2 D Arrays

- Elements in the first row are placed followed by elements of second row and so on
- Contiguous memory allocation is done and address of first byte of memory is stored in the name of the array
- Address of nth element in an array named as a (i.e.) a[n], is determined as: (a+n*b) where b is the number of bytes allocated for the data type

Initialization of a Character 2 D Arrays

```
char tictac[3][3] = { {' ', ' ', ' '}, {' ', ' ', ' '},
                      {' ', ' ', ' '} };
```

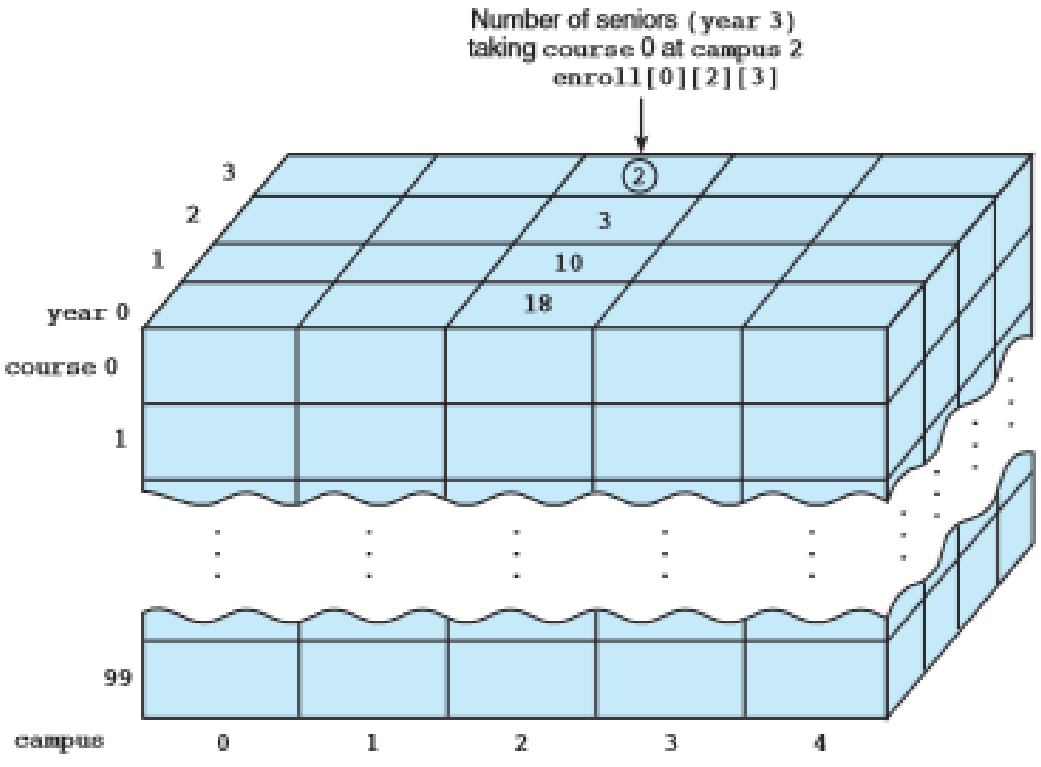
Multi dimensional Arrays

- store the enrolment data for a college
- assume that the college offers 100 (MAXCRS) courses at five different campuses
- Students from any of the four years can opt for the courses

Arrays with Several Dimensions

```
int enroll[MAXCRS][5][4];
```

course *campus* *year*



```c
#include<stdio.h>
void main()
{
        int tictactoe[3][3];
        int i_Counter = 0, j_Counter=0,empty_Cells=0,win = 0;
        //Read the state of the board
        for(i_Counter=0;i_Counter<3;i_Counter++)
        for(j_Counter=0;j_Counter<3;j_Counter++)
                scanf("%d",&tictactoe[i_Counter][j_Counter]);
        //Count the number of empty cells
        for(i_Counter=0;i_Counter<3;i_Counter++)
        for(j_Counter=0;j_Counter<3;j_Counter++)
                if(tictactoe[i_Counter][j_Counter]==-1)
                        empty_Cells++;
        if (empty_Cells==9)
                printf("Initial State");
```

```c
else
{
        //Check rowwise
        for(i_Counter=0;i_Counter<3;i_Counter++)
        {
                if (tictactoe[i_Counter][0]==1&&tictactoe[i_Counter][1]==1&&tictactoe[i_Counter][2]==1)
                        {
                        printf("Player1 Wins");
                        win =1;
                        }
                else if (tictactoe[i_Counter][0]==2&&tictactoe[i_Counter][1]==2&&tictactoe[i_Counter][2]==2)
                        {
                        printf("Player2 Wins");
                        win =1;
                        }
```

```
        }
//Check Columns
else if (tictactoe[0][i_Counter]==1&&tictactoe[1][i_Counter]==1&&tictactoe[2][i_Counter]==1)
        {
        printf("Player1 Wins");
        win =1;
        }
else if (tictactoe[0][i_Counter]==2&&tictactoe[1][i_Counter]==2&&tictactoe[2][i_Counter]==2)
        {
        printf("Player2 Wins");
        win =1;
        }
```

```
}
//Check Diagonal
        if ((tictactoe[0][0]==1&&tictactoe[1][1]==1&&tictactoe[2][2]==1)||
                (tictactoe[0][2]==1&&tictactoe[1][1]==1&&tictactoe[2][0]==1))
                {
                printf("Player1 Wins");
                win =1;
                }
        else if ((tictactoe[0][0]==2&&tictactoe[1][1]==2&&tictactoe[2][2]==2)||
                (tictactoe[0][2]==2&&tictactoe[1][1]==2&&tictactoe[2][0]==2))
                {
                printf("Player2 Wins");
                win =1;
                }
        //Board not empty and no one Wins
        else if(empty_Cells==0)
                printf("Draw");
        else
                printf("Intermediate");
```


janaki@janaki-HP-248-G1:~$ ./a.out
1
2
1
2
1
2
2
1
-1
Intermediatejanaki@janaki-HP-248-G1:~$ ./a.out
1
2
1
2
1
2
2
1
2
Drawjanaki@janaki-HP-248-G1:~$ gedit tictactoe.c

Functions

Types

- User Defined Functions
  - With Arguments With Return Value
  - Without Arguments With Return Value
  - With Arguments Without Return Value
  - Without Arguments Without Return Value
- Built-in Functions
  - String Functions
  - Mathematical Functions etc.

Three components of Functions

- Function Prototype / Function Declaration
- Function Definition
- Function Call / Function Invoke

Function Prototype

```
return_type function_name(datatype1,
                          datatype2, datatype3…..)
```

Function Definition

```
return_type function_name(data_type1 arg1,
        data _type2 arg2, ……) {
                local declaration;
                executable statements;
                return statement;
}
```

Function Call

variable = function_name(para1, para2,……);

```
//Call by Value Example - Swapping 2 numbers using Call by Value
#include <stdio.h>


  void swap(int, int);


  int main()
  {
     int x, y;


     printf("Enter the value of x and y\n");
     scanf("%d%d",&x,&y);


     printf("Before Swapping\nx = %d\ny = %d\n", x, y);


     swap(x, y);


     return 0;
  }


  void swap(int a, int b)
  {
     int temp;


     temp = b;
     b = a;
     a = temp;
     printf("Values of x and y after swapping are %d  %d\n",a,b);
```

```
}
```

The **call by reference** method of passing arguments to a function 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. It means the changes made to the parameter affect the passed argument.

To pass a value by reference, argument pointers are passed to the functions just like any other value. So accordingly you need to declare the function parameters as pointer types as in the following function **swap()**, which exchanges the values of the two integer variables pointed to, by their arguments.

```
/* 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;
}
```

Let us now call the function **swap()** by passing values by reference as in the following example −

```
#include <stdio.h>

/* function declaration */
void swap(int *x, int *y);

int main () {

   /* local variable definition */
   int a = 100;
   int b = 200;

   printf("Before swap, value of a : %d\n", a );
   printf("Before swap, value of b : %d\n", b );

   /* calling a function to swap the values.
      * &a indicates pointer to a ie. address of variable a and
      * &b indicates pointer to b ie. address of variable b.
   */
   swap(&a, &b);

   printf("After swap, value of a : %d\n", a );
   printf("After swap, value of b : %d\n", b );

   return 0;
}
```

Functions

Type-I With Arguments and With Return Values

```
main() {

    int add(int, int);

    int a, b, c;

    printf("\t\t\tFunctions\n");

    printf("\t\t\t---------\n");

    printf("\t\t\tType-I\n");

    printf("\t\t\t------\n");

    printf("\tWith Arguments and With Return Value\n");

    printf("\t-------------------------------------\n");

    printf("Enter the numbers a and b:\n");

    scanf("%d %d", &a, &b);

    c = add(a, b);

    printf("The Result is : %d\n",c);

    getch();

    }

int add(int x, int y) {

  int z;

  z = x + y;

  return z;

}
```

Type-II Without Arguments and With Return Values

```
main() {

    int add();

    int c;

    printf("\t\t\tFunctions\n");

    printf("\t\t\t---------\n");

    printf("\t\t\tType-II\n");
```

```
    printf("\t\t\t-------\n");
    printf("\tWithout Arguments and With Return Value\n");
    printf("\t---------------------------------------\n");
    c = add();
    printf("The Result is : %d\n",c);
    getch();
    }
int add() {
  int x, y, z;
  printf("Enter the numbers x and y:\n");
  scanf("%d %d", &x, &y);
  z = x + y;
  return z;
}
```

Type-III With Arguments and Without Return Values

```
#include <stdio.h>
main() {
    void add(int, int);
    int a, b;
    printf("\t\t\tFunctions\n");
    printf("\t\t\t---------\n");
    printf("\t\t\tType-III\n");
    printf("\t\t\t--------\n");
    printf("\tWith Arguments and Without Return Value\n");
    printf("\t---------------------------------------\n");
    printf("Enter the numbers a and b:\n");
    scanf("%d %d", &a, &b);
    add(a, b);
    getch();
    }
void add(int x, int y) {
  int z;
  z = x + y;
  printf("The Result is : %d\n",z);
}
```

Type-IV Without Arguments and Without Return Values

```
main() {
    void add();
    int a, b;
    printf("\t\t\tFunctions\n");
    printf("\t\t\t---------\n");
    printf("\t\t\tType-IV\n");
    printf("\t\t\t-------\n");
    printf("\tWithout Arguments and Without Return Value\n");
    printf("\t------------------------------------------\n");
    add();
    getch();
    }
void add() {
  int a, b, c;
  printf("Enter the numbers a and b:\n");
  scanf("%d %d", &a, &b);
  c = a + b;
  printf("The Result is : %d\n",c);

}
```

Function Recursion

Fibonacci Series

```
#include <stdio.h>

int fibonacci(int i) {

   if(i == 0) {
      return 0;
   }

   if(i == 1) {
      return 1;
   }
   return fibonacci(i-1) + fibonacci(i-2);
}

int  main() {

   int i;

   for (i = 0; i < 10; i++) {
      printf("%d\t\n", fibonacci(i));
   }

   return 0;
}
```

Factorial of a given number

```
#include <stdio.h>

long int multiplyNumbers(int n);

int main()

{

    int n;

    printf("Enter a positive integer: ");

    scanf("%d", &n);

    printf("Factorial of %d = %ld", n, multiplyNumbers(n));

    return 0;

}

long int multiplyNumbers(int n)









{

    if (n >= 1)

        return n*multiplyNumbers(n-1);

    else

        return 1;

}
```