Hello CPPBuddies

DAY-11

Hello CPPBudddies

Day 11

Welcome

To

C++ COMPLETE BOOTCAMP

Your Guide To A Solid Foundataion in C++

Let us begin

WELCOME MEMBERS

C++ Complete Bootcamp

YOUR GUIDE TO PROGRAMMING

In association with

Inspire Club, MANIT BHOPAL

Inline functions

The use of functions provides many benefits, including:

- The code inside the function can be **reused**.
- It is much easier to change or update the code in a **function** (which needs to be done once) than for every in-place instance. Duplicate code is a recipe for inefficiency and errors.
- It makes your **code easier to read and understand**, as you do not have to know how a function is implemented to understand what it does (assuming responsible function naming or comments).
- Functions make your program easier to **debug**.

However, one major **downside** of functions is that every time a function is called, there is a certain amount of **performance overhead** that occurs.

Function Call => STACK FRAME CREATION

This is because the CPU must store the address of the current instruction it is executing (so it knows where to return to later) along with other registers,
all the function parameters must be created and assigned values,
and the program has to branch to a new location

For functions that are large and/or perform complex tasks, the overhead of the function call is usually insignificant compared to the amount of time the function takes to run.

However, for small, commonly-used functions, the time needed to make the function call is often a lot more than the time needed to actually execute the function's code.

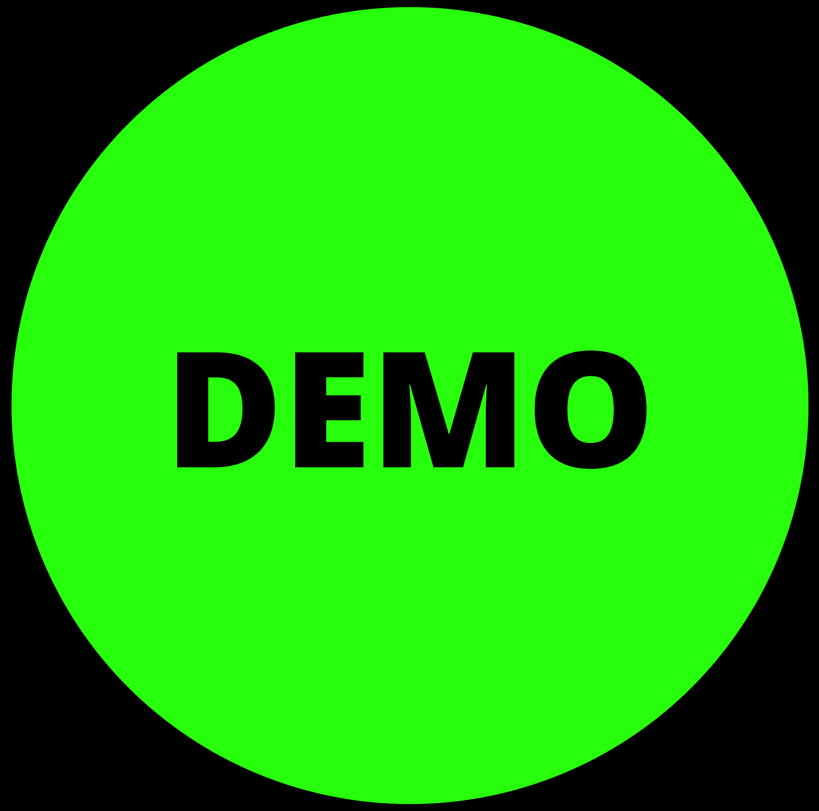
This can result in a substantial performance penalty.

C++ offers a way to combine the advantages of functions with the speed of code written in-place: **inline functions**.

The **inline keyword** is used to request that the compiler treat your function as an inline function.

Code written in-place is significantly faster.

When the compiler compiles your code,
all inline functions are **expanded in-place** --
that is, the function call is replaced with a **copy of the contents of the function** itself,
which removes the function call overhead!



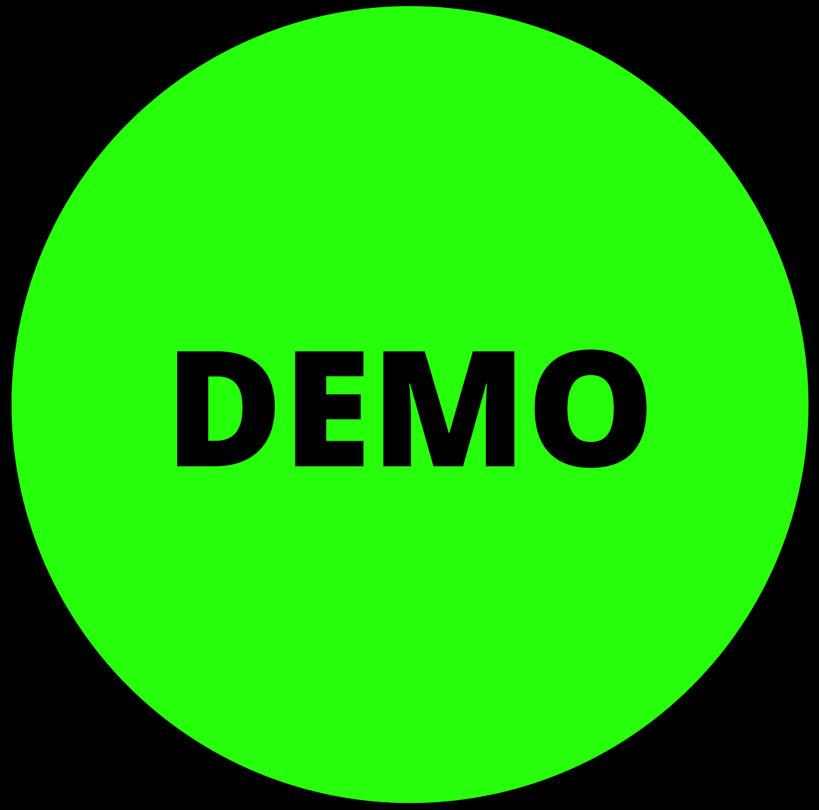
Inline Function

Macros:

Macros are a piece of code in a program which is given some name.

Whenever this name is encountered by the compiler, the compiler replaces the name with the actual piece of code.

The ‘**#define**’ directive is used to define a macro.



Macros

Note: There is no semi-colon(“;”) at the end of macro definition.

Macro definitions do not need a semi-colon to end.

Macros with arguments:

We can also pass arguments to macros.

Macros defined with arguments works similarly as functions.

Problem No. 1

Write a macro that is used for finding the multiplicative inverse of a number

Problem No. 2

Write a C++ Program for finding the **biggest** of two numbers

ASCII Values

ASCII

American Standard Code for Information Interchange

each character => numerical value

This is called ASCII Value of that character

ASCII Table

Dec = Decimal Value
Char = Character

'5' has the int value 53
if we write '5'-'0' it evaluates to 53-48, or the int 5
if we write char c = 'B'+32; then c stores 'b'

Dec	Char	Dec	Char	Dec	Char	Dec	Char
0	NUL (null)	32	SPACE	64	@	96	`
1	SOH (start of heading)	33	!	65	A	97	a
2	STX (start of text)	34	"	66	B	98	b
3	ETX (end of text)	35	#	67	C	99	c
4	EOT (end of transmission)	36	$	68	D	100	d
5	ENQ (enquiry)	37	%	69	E	101	e
6	ACK (acknowledge)	38	&	70	F	102	f
7	BEL (bell)	39	'	71	G	103	g
8	BS (backspace)	40	(	72	H	104	h
9	TAB (horizontal tab)	41	)	73	I	105	i
10	LF (NL line feed, new line)	42	*	74	J	106	j
11	VT (vertical tab)	43	+	75	K	107	k
12	FF (NP form feed, new page)	44	,	76	L	108	l
13	CR (carriage return)	45	-	77	M	109	m
14	SO (shift out)	46	.	78	N	110	n
15	SI (shift in)	47	/	79	O	111	o
16	DLE (data link escape)	48	0	80	P	112	p
17	DC1 (device control 1)	49	1	81	Q	113	q
18	DC2 (device control 2)	50	2	82	R	114	r
19	DC3 (device control 3)	51	3	83	S	115	s
20	DC4 (device control 4)	52	4	84	T	116	t
21	NAK (negative acknowledge)	53	5	85	U	117	u
22	SYN (synchronous idle)	54	6	86	V	118	v
23	ETB (end of trans. block)	55	7	87	W	119	w
24	CAN (cancel)	56	8	88	X	120	x
25	EM (end of medium)	57	9	89	Y	121	y
26	SUB (substitute)	58	:	90	Z	122	z
27	ESC (escape)	59	;	91	[	123	{
28	FS (file separator)	60	<	92	\	124	|
29	GS (group separator)	61	=	93	]	125	}
30	RS (record separator)	62	>	94	^	126	~
31	US (unit separator)	63	?	95	_	127	DEL

Some Common ASCII VALUES

A => 65

B => 66

……..

a => 97

b => 98

……..

0 => 48

1 => 49

….

EXAMPLE

Print the ASCII values of:

A--Z

a--z

0--9

Let us practice some problems

Example

Check whether a character is alphabet, digit or special character

Basic C programming, Relational operators, Logical operators, If else

- A character is alphabet if it in between a-z or A-Z.
- A character is digit if it is in between 0-9.
- A character is special symbol character if it neither alphabet nor digit.

Example

Input

Input any character: 3

Output

3 is digit

Decode Problem

ACE => 135

BDC123 => 243272829

12 => 2728

abd => 373840

1Ac2 => 2713928

Highest ASCII Character

Input N characters

Find the character with highest ASCII Value

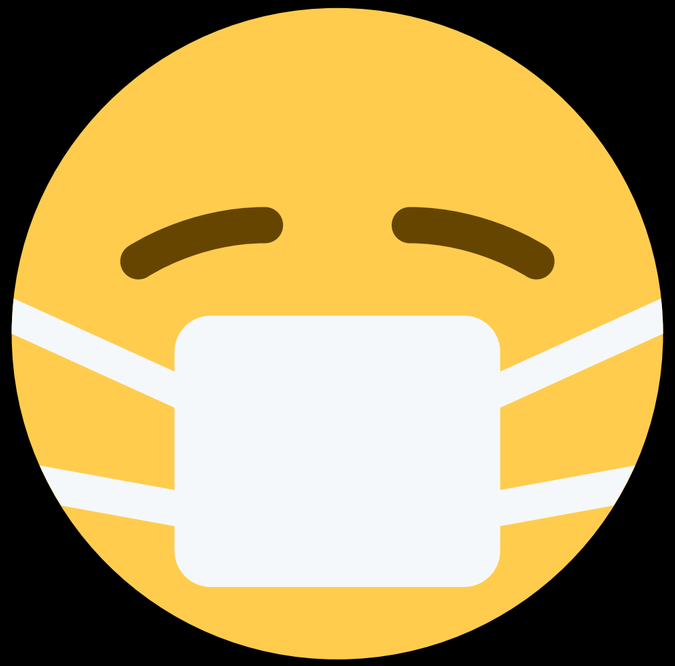
SUM of ASCII

Input a sequence of characters and find the sum of all ASCII values

Number System

THANK YOU

keep calm,
wear mask,
and
study hard



ISSUED IN PUBLIC INTEREST

whoami

AKASH MAJI

Your Mentor