CS101-2020
Lec 5

Take a Plunge into the Sea of *Cs*

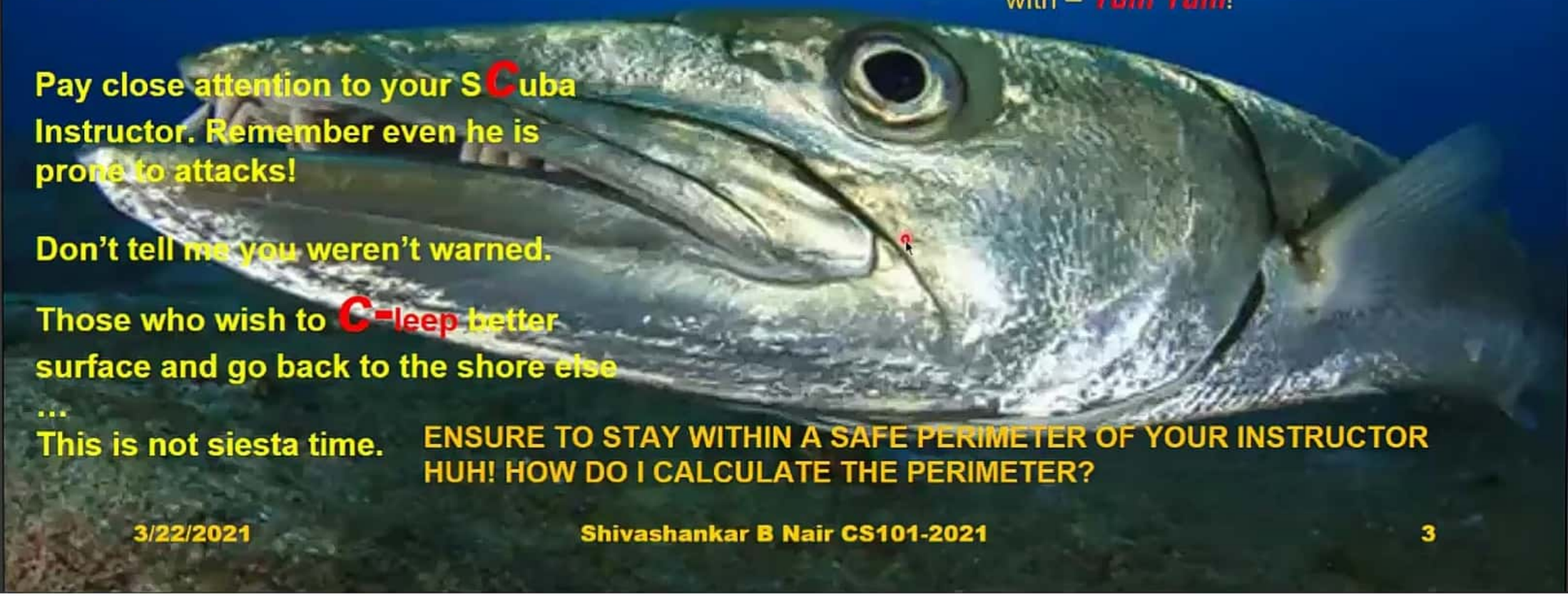
3/22/2021

We are now in the reefs of the Sea of

Cs infested by the deadly

BARRACUDAs. Danger lurks everywhere!

Barracudas, (Cuda), are large, predatory, ray-finned fish whose appearance is as fearsome as their ferocious behaviour.
They can zip past you at speeds of around 50-58 kmph and while doing so even take a bite of you along with – *Yum Yum*!



Pay close attention to your SCuba Instructor. Remember even he is prone to attacks!

Don't tell me you weren't warned.

Those who wish to C=leep better surface and go back to the shore else …

This is not siesta time.

ENSURE TO STAY WITHIN A SAFE PERIMETER OF YOUR INSTRUCTOR
HUH! HOW DO I CALCULATE THE PERIMETER?

FINDING THE PERIMETER IN THE SEA OF Cs

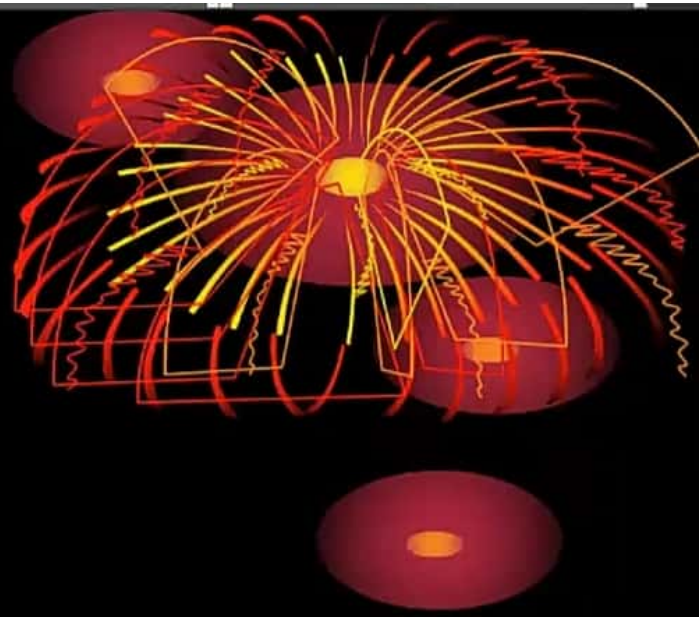
```
/* This program finds the area & perimeter
   of a rectangle */

#include <stdio.h>

main()
{
   int l, b, area, perimeter;
   b = 4;
   l = 6;
   area = l*b;
   perimeter = 2*(l+b);
   printf("Area = %d \n", area);
   printf ("Perimeter = %d \n", perimeter);
}
/* End of main  */
```

Don't forget the semi colon at the end of each statement! But not so for the #include

Note the blank spaces.

Will cause the value within the variable 'area' to be printed here in decimal form.
\n will cause the cursor to be moved to the next line.

Compilation & the Output



- Compiler ignores comment lines
- `#include` line **must not** end with semicolon
- Every statement is terminated by semicolon

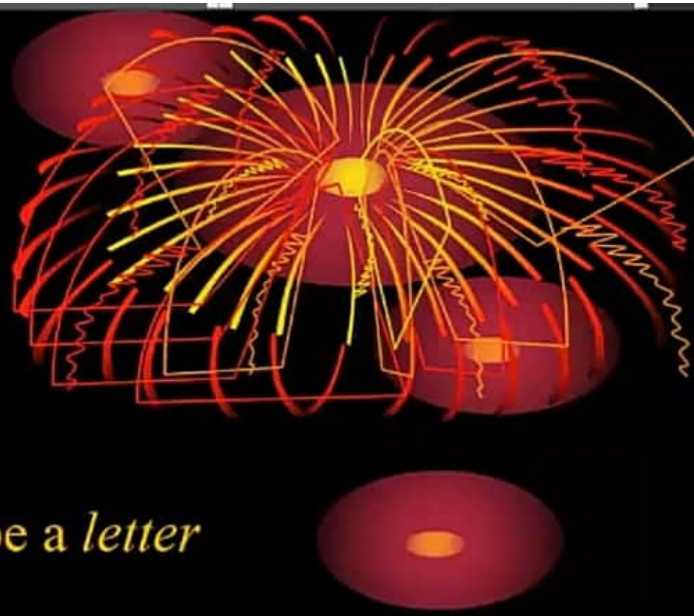
Compile program perimeter.c
Execute the program

Output of the program

```
Area = 24
Perimeter = 20
```

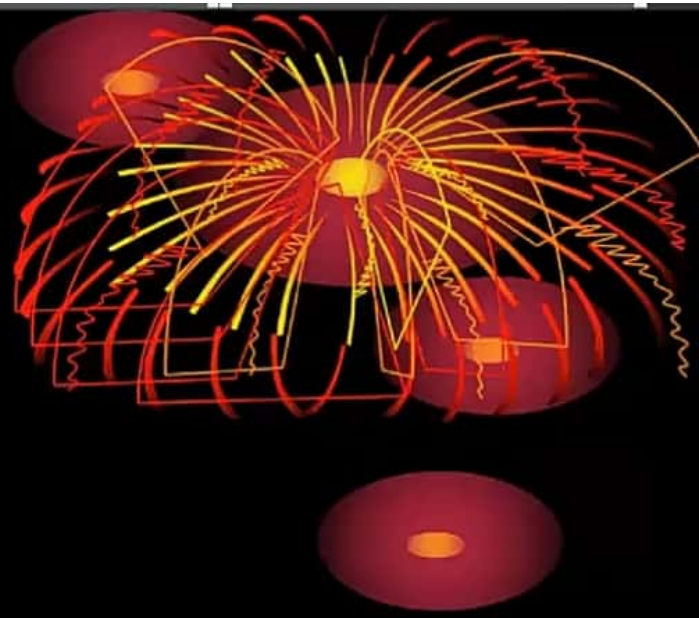
NB: There are blank spaces b'cos we specified so in the printf statement.

Variables and Constants



- Form the basic data objects manipulated in a program
- *Variables* should have a *name* (*identifier*) and a *value*
- *Names* comprise *letters* and *digits*; the first character must be a *letter*
- E.g.: *total, n, sum1, sum2*
- "_" (underscore) is also a *letter*.
- E.g.: *val_1* is a valid name.
- **DO NOT** use variable names that begin with an underscore, since library functions often use such names.

Names (Identifiers) ...



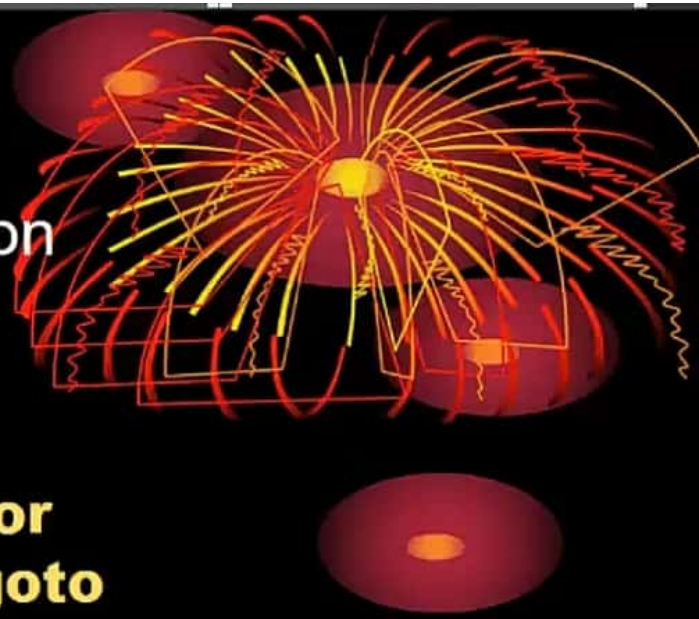
- Upper case and lower case letters are distinct (Case sensitive).
- Thus *x* and *X* *are two different letters*.
  - Example:

    area
    Area
    AREA
    ArEa

    are all seen as different variables by the compiler.

- *C* has reserved words like:
  - *if, else, int, float,…* which should **NOT** be used as variable names.

C Keywords / Reserved Words

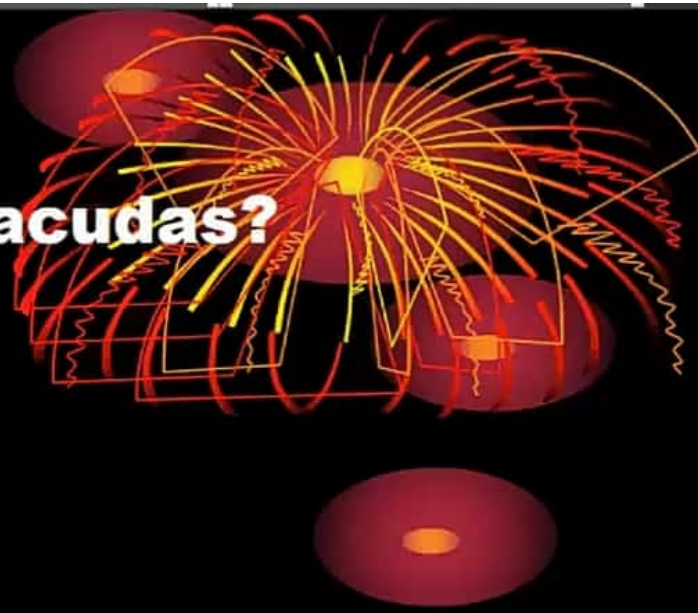


- Reserved words are not available for redefinition
- Cannot be used as variable name
- Have special meaning in *C*

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

Fun with danger: Raise your adrenalin by *Baiting the Barracudas*

Which variable names will attract the barracudas?



- secondName
- 2ndName 
- %Marks 
- char 
- charAndNum
- _addNumber

Variable Name Examples

Correct	Wrong
secondName	2ndName /* starts with a digit */
_addNumber	%Marks /* contains invalid character %*/
charAndNum	char /* reserved word */

Identifier in C: Variable Name Examples

Correct	Barracuda Bait?
`annual_rate`	
`stage4mark`	
`annual rate`	 Contains a space
`my\nName`	 Contains new line character \n

Names ... What about these?

- *$total* → $ sign is not allowed
- *1st* → *Begins with a no.*
- *no.* → *Has a period at the end*
- *case* → *Is a reserved word*
- *_total* → *Has an _ in the beginning (avoid)*
- *end* → *Is a reserved word*

Which of these identifiers/variables are correct?

AREA

area_under_the_curve

3D

num45

#values

x_yt3

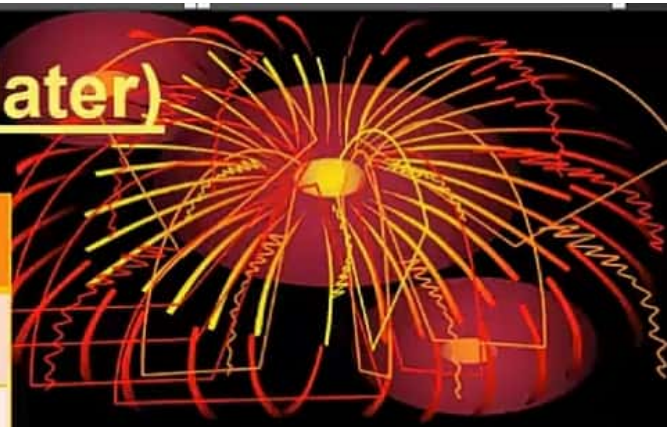
pi

num$

%done

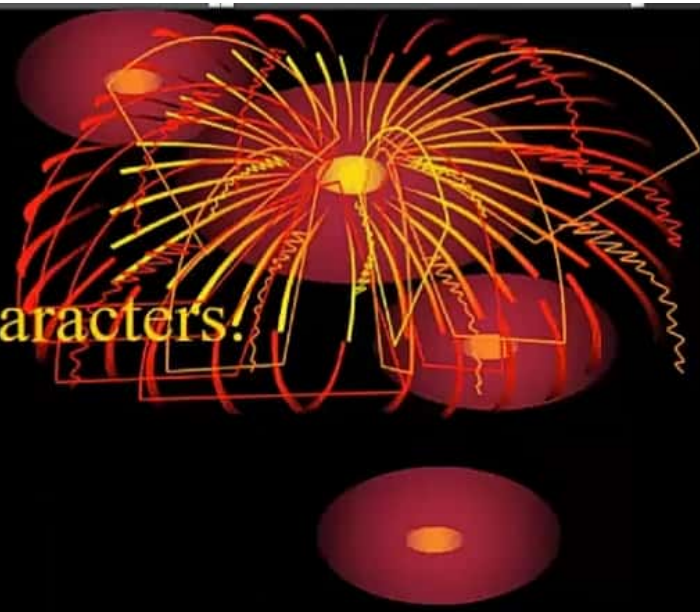
lucky***

C Variables: Examples (More about types later)



Correct
`int x, y, z, my_data = 4;`
`short number_one;`
`long Cost_of_Car;`
`unsigned int positive_number;`
`char Gender;`
`float commission, yield = 4.52;`
`char the_initial = 'M'; // A char`
`char cntryName[20]="India";// A string`

Length of a name

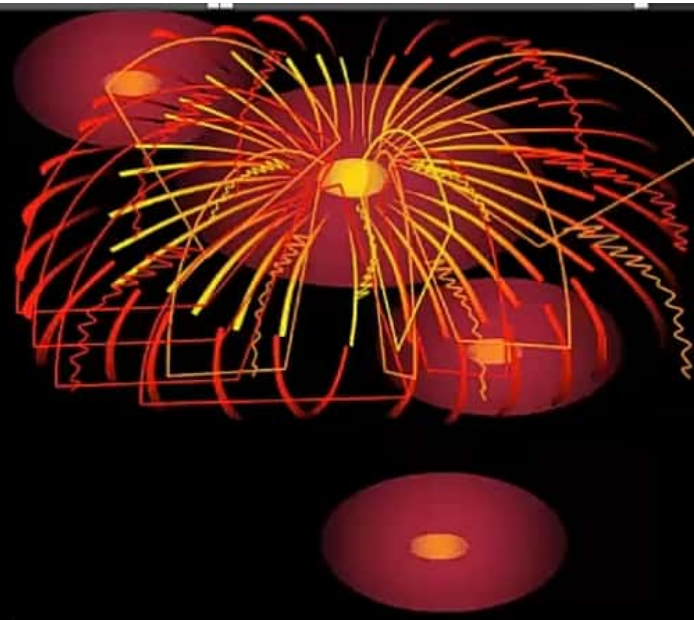


- ANSI C is said to discriminate up to 31 characters!
- Could be more in some compilers!
- Point is not the length but the name/identifier should be reflect its inherent meaning to make the program comprehensible (*Elegant programming style*)
- E.g. *length, base, perimeter_of_rectangle* clearly specify what the identifier represents

The #define preprocessor directive or Macro

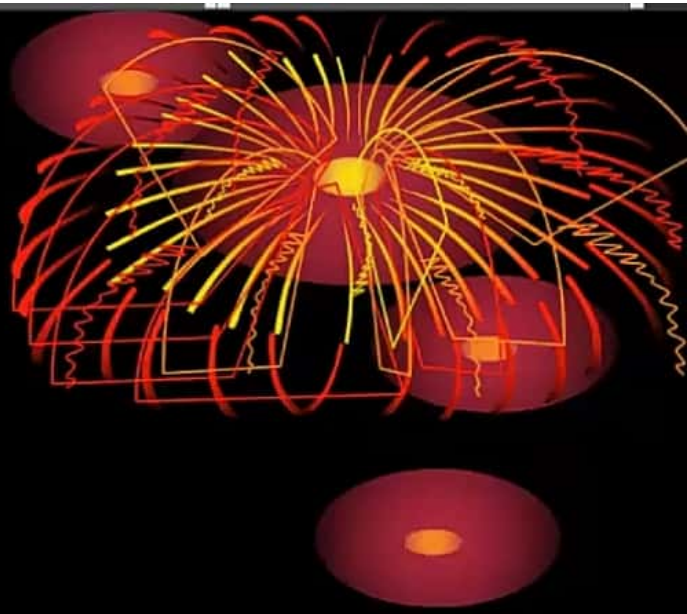
- An easier way to save typing/writing in a program when a variable or constant is referred frequently in the body of the program is to use #define

#define ...



- Use all uppercase for symbolic constants
- Examples:
- #define PI 3.14159 NB: No semicolon at the end
- #define AGE 52
- #define LINK "iitg.c.net"
- In the body of the program you can now use PI instead of 3.14159.

Using #define



```
#include <stdio.h>

#define NAME "Indian Institute of Technology Guwahati"

#define AGE 26

int main()
{
    printf("The %s is %d years old.\n", NAME, AGE);
    return 0;
}
```

String

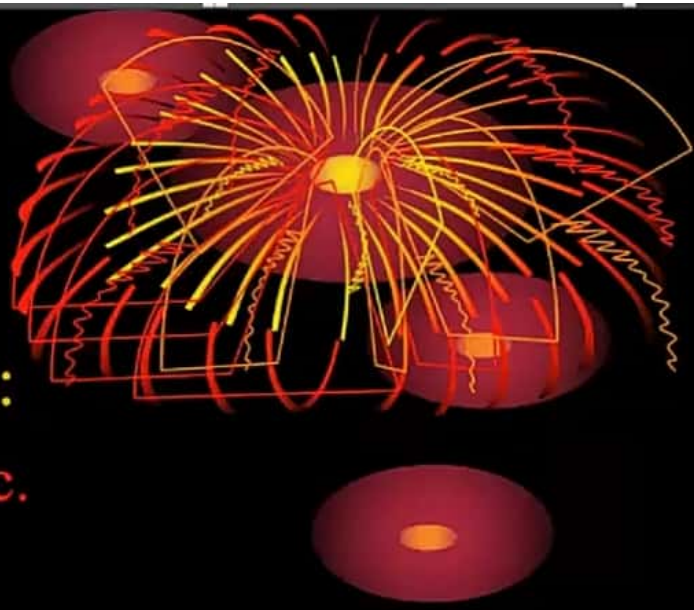
Integer

String Integer

Output:

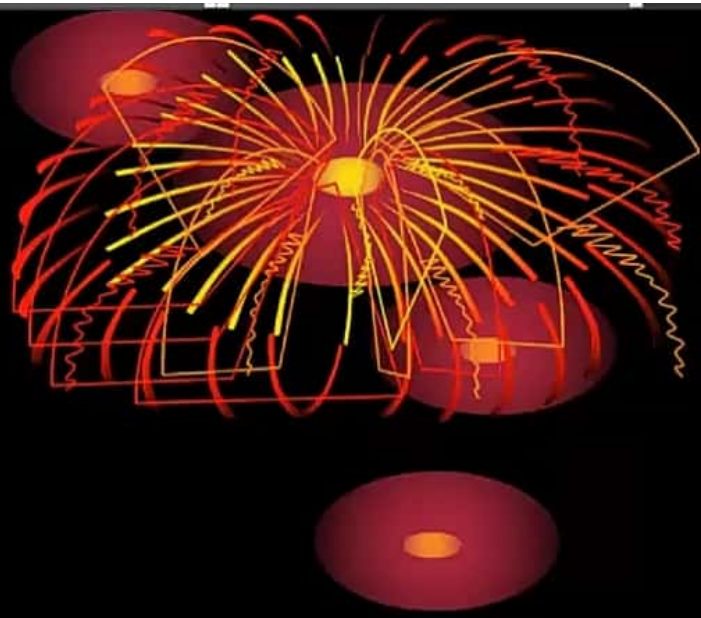
The Indian Institute of Technology Guwahati is 26 years old.

Data Types



- There are various categories of data items:
  - *integers*, *real numbers*, *character*, *strings*, etc.
- Each category is called a *type*.
- There are inbuilt types (basic types) and user-defined types.

Declarations ...



- Syntax for a declaration is:

  type var_1, var_2, ... , var_n;

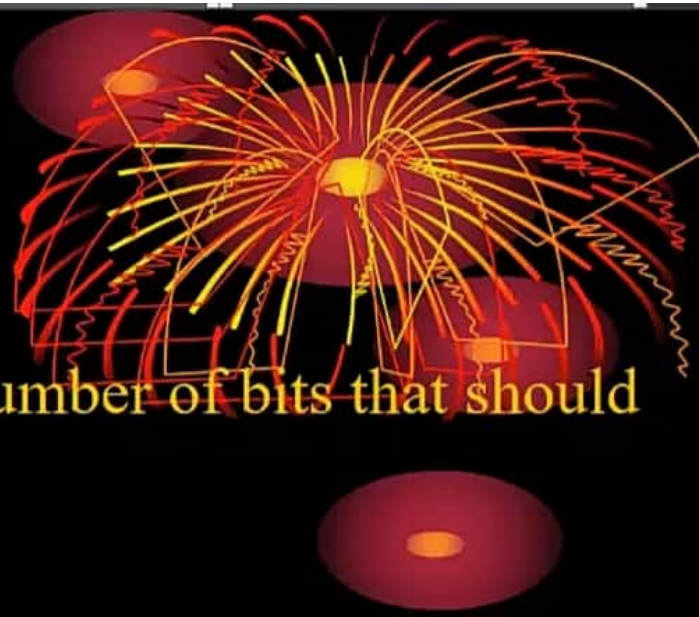
```
int i;
int j;
int k;
```

This may also be written as -

```
int  i, j, k;
```

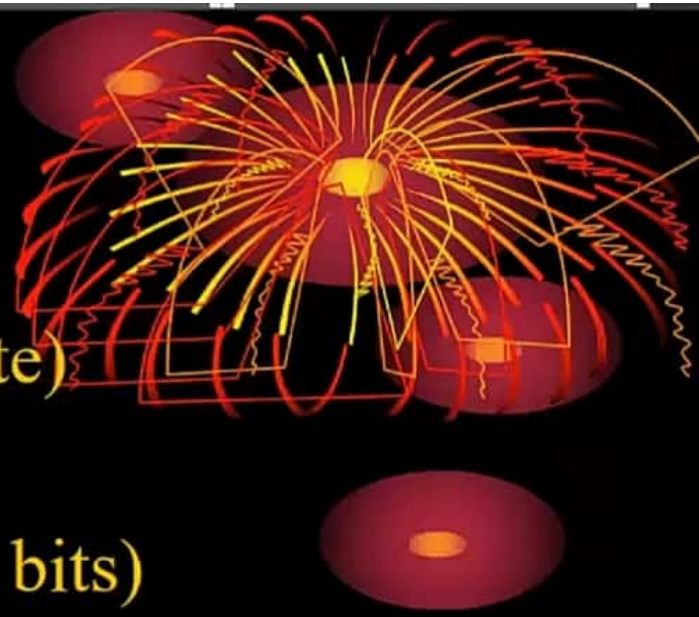
- All user-defined variables should be declared initially.

Declarations ...



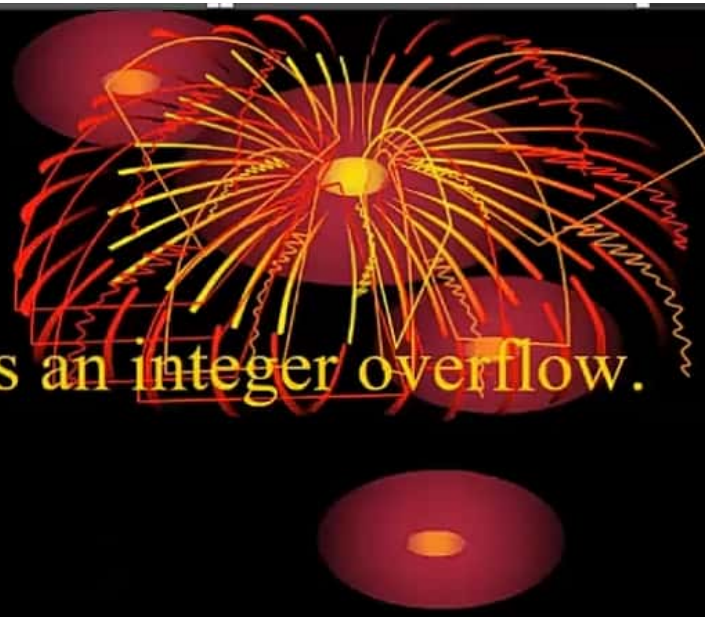
- Declarations specify type and also the size (the number of bits that should be allocated to store the declared variable).
- These sizes are machine (Hardware) dependent.
- Size determines the range of values you can use.

Basic data types



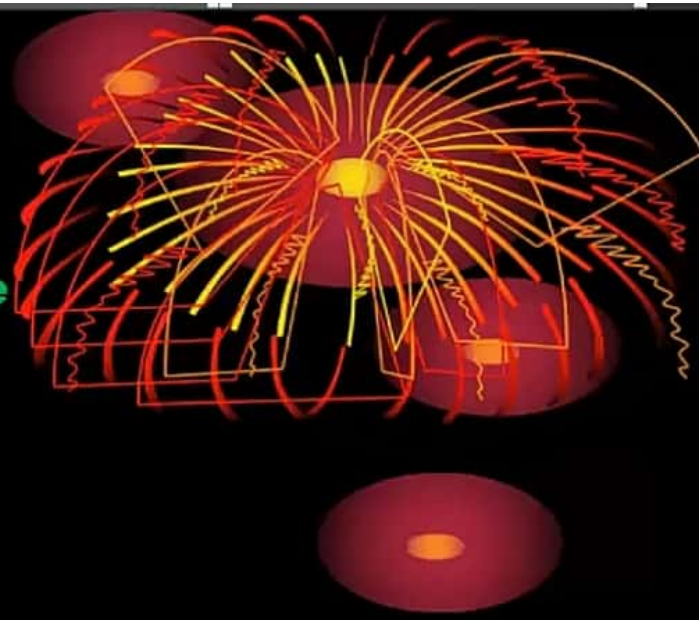
- character (*char*) → single character (1 byte)
- integer (*int*) → an integer number (32 bits)
- real (*float*) → a real valued number (4 bytes)
  (*double*) → a real valued number with higher precision (8 bytes)

Overflow



- If the number goes beyond these values it is an integer overflow.
- This may cause logically incorrect results.
- The ***onus/responsibility is on the programmer*** to keep the number within range.

How to find the size?

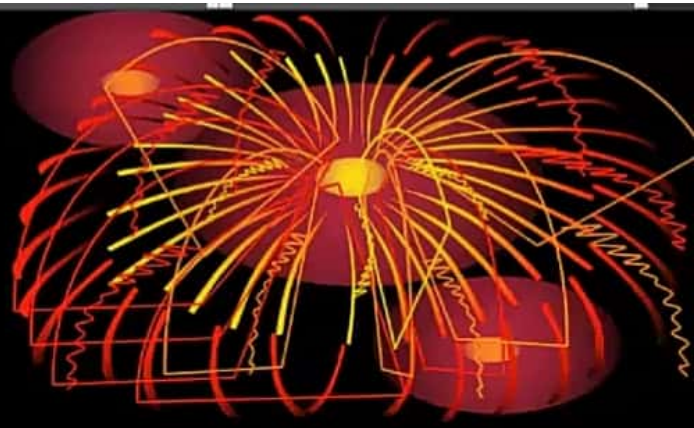


- ***sizeof()*** **is a function which gives you the size of the memory (in bytes) used by an entity.**

E.g.:

```
#include<stdio.h>
main()
{
  int s;
  s = sizeof( int );
  printf("The size of an int is: %d", s);
}
```

Try this...



- Find the size allocated for other types –
  - char, int, long, unsigned, ...

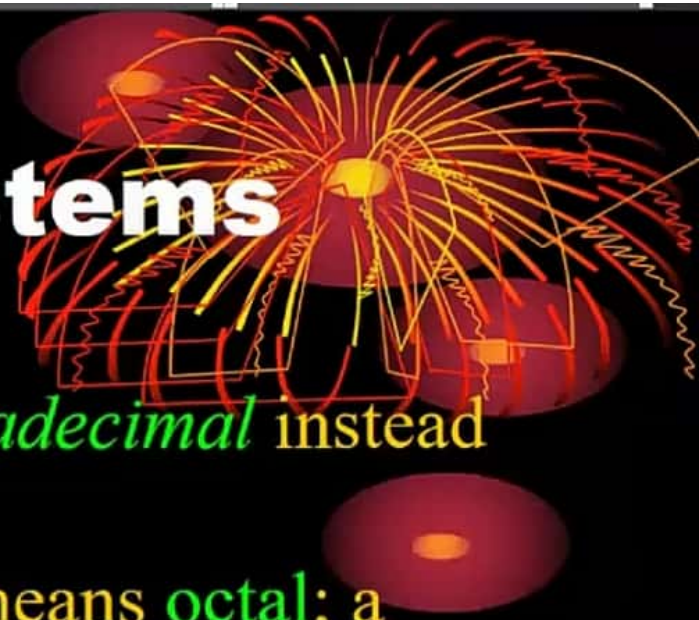
Testing size of Numeric Data

```
#include<stdio.h>
int main(){
  printf("size of char %d\n", sizeof(char));                    //1
  printf("size of short %d\n",sizeof(short));                   //2
  printf("size of int %d\n",sizeof(int));                       //4
  printf("size of long int %d\n",sizeof(long int));             //8
  printf("size of float \n",sizeof(float));                     //4
  printf("size of double %d\n",sizeof(double));                 //8
  printf("size of long double %d\n", sizeof(long double));      //16
  return 0;
}
```

EEK!, Barracuda attack! Did we do something wrong for it to detect our presence?

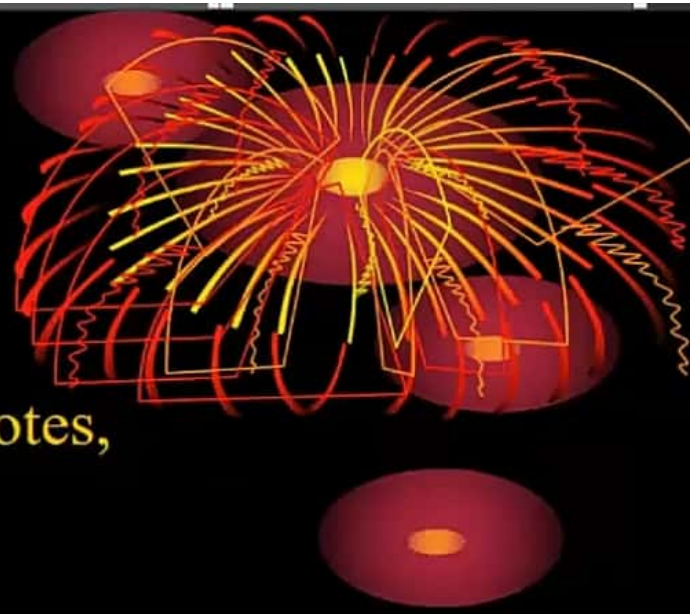
```
#include<stdio.h>
int main(){
  printf("size of char %d\n", sizeof(char));                    //1
  printf("size of short %d\n",sizeof(short));                   //2
  printf("size of int %d\n",sizeof(int));                       //4
  printf("size of long int %d\n",sizeof(long int));             //8
  printf("size of float %d\n",sizeof(float));                                  //4
  printf("size of double %d\n",sizeof(double));                 //8
  printf("size of long double %d\n", sizeof(long double));      //16
  return 0;
}
```

Specifying some no. systems

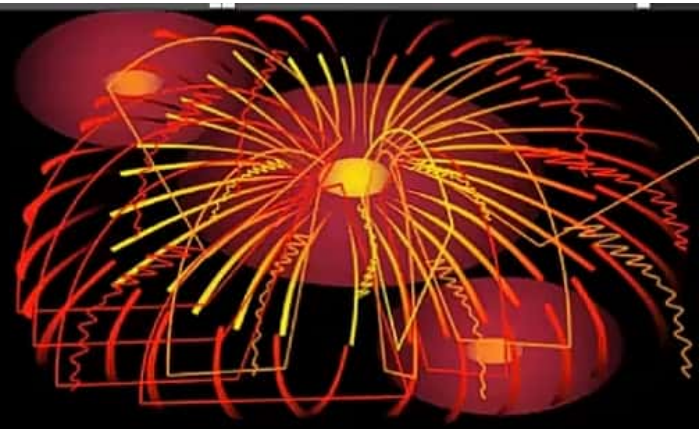


- An integer can be specified in *octal* or *hexadecimal* instead of decimal.
- A leading *0* (zero) on an integer constant means octal; a leading *0x* or *0X* means hexadecimal.
- Eg: decimal 31 = 037 (octal)
  = 0x1F
  = 0X1F (hexa)

Constants ...



- A character constant is written within a single quotes,
  - e.g. 'x'
- Actually a character is stored as an integer.
- The integer value of a character is often its ASCII value.
- (*In American Standard Code for Information Interchange - ASCII character set each character is given an integer value.*)
- '0' → character zero → ASCII value is 48
- '0' is unrelated to the numeric value 0



ASCII control characters (character code 0-31)
The first 32 characters in the ASCII-table are non-printable control codes and are used to control peripherals such as printers.

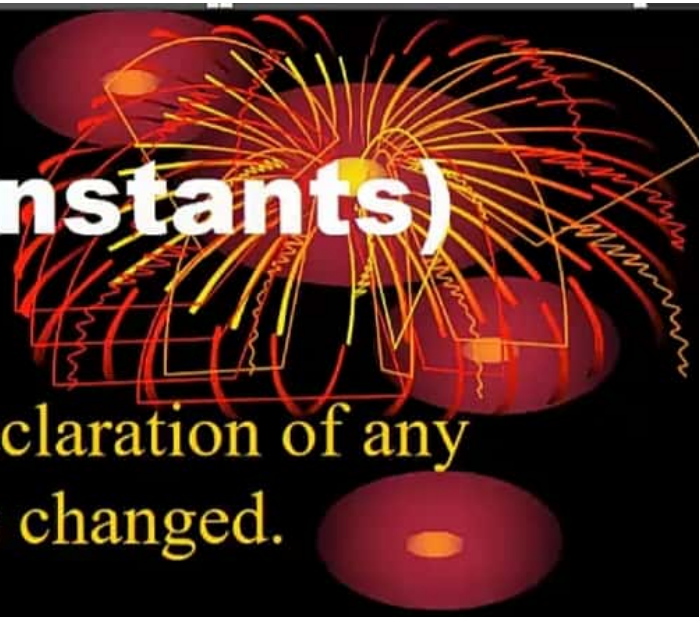
DEC	OCT	HEX	BIN	Symbol	HTML Number	HTML Name	Description
0	000	00	00000000	NUL	&#000;		Null char
1	001	01	00000001	SOH	&#001;		Start of Heading
2	002	02	00000010	STX	&#002;		Start of Text
3	003	03	00000011	ETX	&#003;		End of Text
4	004	04	00000100	EOT	&#004;		End of Transmission
5	005	05	00000101	ENQ	&#005;		Enquiry
6	006	06	00000110	ACK	&#006;		Acknowledgment
7	007	07	00000111	BEL	&#007;		Bell
8	010	08	00001000	BS	&#008;		Back Space
9	011	09	00001001	HT			Horizontal Tab
10	012	0A	00001010	LF
	Line Feed
11	013	0B	00001011	VT	&#011;		Vertical Tab
12	014	0C	00001100	FF			Form Feed
13	015	0D	00001101	CR
	Carriage Return
14	016	0E	00001110	SO	&#014;		Shift Out / X-On
15	017	0F	00001111	SI	&#015;		Shift In / X-Off
16	020	10	00010000	DLE	&#016;		Data Line Escape
17	021	11	00010001	DC1	&#017;		Device Control 1 (oft. XON)
18	022	12	00010010	DC2	&#018;		Device Control 2
19	023	13	00010011	DC3	&#019;		Device Control 3 (oft. XOFF)
20	024	14	00010100	DC4	&#020;		Device Control 4
21	025	15	00010101	NAK	&#021;		Negative Acknowledgement
22	026	16	00010110	SYN	&#022;		Synchronous Idle
23	027	17	00010111	ETB	&#023;		End of Transmit Block
24	030	18	00011000	CAN	&#024;		Cancel
25	031	19	00011001	EM	&#025;		End of Medium
26	032	1A	00011010	SUB	&#026;		Substitute

46	056	2E	00101110	.	&#46;		Period, dot or full stop
47	057	2F	00101111	/	&#47;		Slash or divide
48	060	30	00110000	0	&#48;		Zero
49	061	31	00110001	1	&#49;		One
50	062	32	00110010	2	&#50;		Two
51	063	33	00110011	3	&#51;		Three
52	064	34	00110100	4	&#52;		Four
53	065	35	00110101	5	&#53;		Five
54	066	36	00110110	6	&#54;		Six
55	067	37	00110111	7	&#55;		Seven
56	070	38	00111000	8	&#56;		Eight
57	071	39	00111001	9	&#57;		Nine
58	072	3A	00111010	:	&#58;		Colon
59	073	3B	00111011	;	&#59;		Semicolon
60	074	3C	00111100	<	&#60;	&lt;	Less than (or open angled bracket)
61	075	3D	00111101	=	&#61;		Equals
62	076	3E	00111110	>	&#62;	&gt;	Greater than (or close angled bracket)
63	077	3F	00111111	?	&#63;		Question mark
64	100	40	01000000	@	&#64;		At symbol
65	101	41	01000001	A	&#65;		Uppercase A
66	102	42	01000010	B	&#66;		Uppercase B
67	103	43	01000011	C	&#67;		Uppercase C
68	104	44	01000100	D	&#68;		Uppercase D
69	105	45	01000101	E	&#69;		Uppercase E
70	106	46	01000110	F	&#70;		Uppercase F
71	107	47	01000111	G	&#71;		Uppercase G

Constants

- Some characters cannot be written normally, hence they have special codes called *escape* sequences.
- '\n' → newline; '\a' → bell; '\\' → backslash
  '\b' → backspace; '\t' → tab; '\0' → null
  '\'' → single quote; '\"' → double quote;
- Explore more escape characters as an exercise.

Constants ... (Named constants)



- The qualifier *const* can be applied to the declaration of any variable to specify that its value will not be changed.

- Eg: *const int* sum = 100;

Phew!

So much for Types, Variables and Constants!

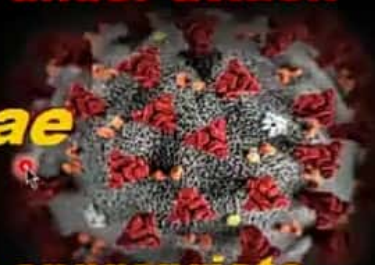
Hope you'll enjoyed the dip in the

Sea of Cs

Now that we have resurfaced, you may take off all sCuba diving gear

BUT DON'T FORGET TO WEAR YOUR MASKS

On the terrestrial side we could be under attack by



OrthoCoronavirinae

aka COVID-19 virus

Play safe, wear a mask, maintain appropriate physical distance and sanitize your hands often

See you'll tomorrow same time!

Until then

Take care

"It's just a hunch, but you're not a certified C diver, are you?"