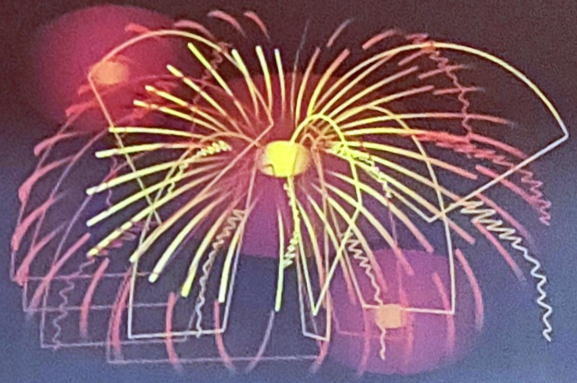
Compilation & the Output



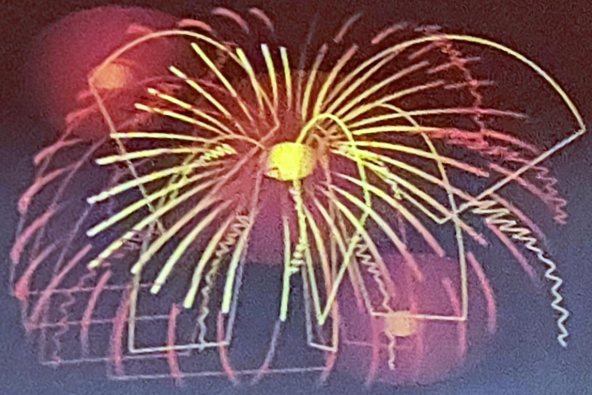
- Compiler ignores comment lines
- `#include` line must not end with semicolon
- Every statement is terminated by semicolon

cc file_name	→ ***compile program perimeter.c***
./a.out	→ ***execute the program***

Output of the program

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

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 treated as 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

} The Compiler treats all these as different variables.

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

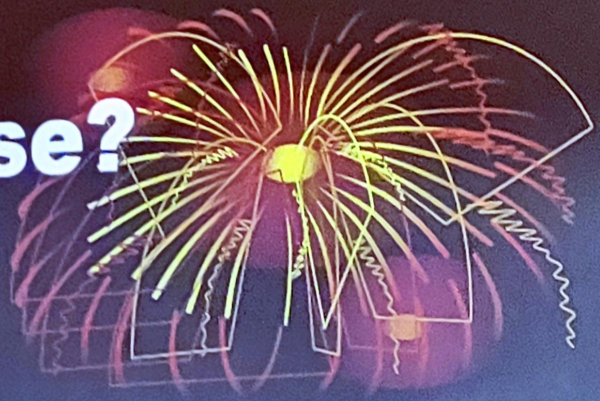
- Reserved words are not available for redefinition
- Cannot be used as a variable name
- Have special meaning in *C*

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

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

Correct	Wrong
annual_rate	annual rate /* contains a space */
stage4mark	my\nName /* contains new line character, \n */

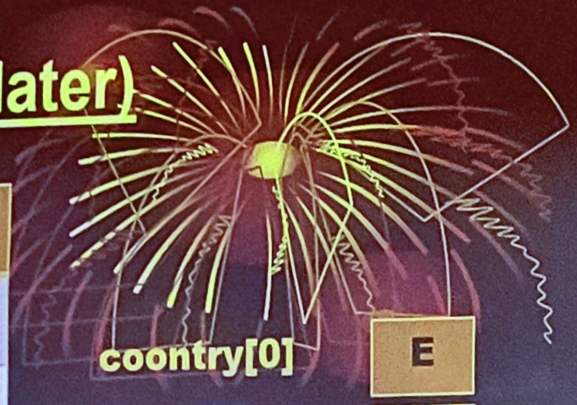
Names ... What about these?



- *$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*

C Variables: Examples (More about types later)

Correct
`int  x, y, z, my_data = 4;`
`short  number_one;`
`long  TypeofCar;`
`unsigned int  positive_number;`
`char  Title;`
`float  commission, yield = 4.52;`
`char the_initial = 'M';  // A char`
`char coontry[20]="Eswatini";// A string`



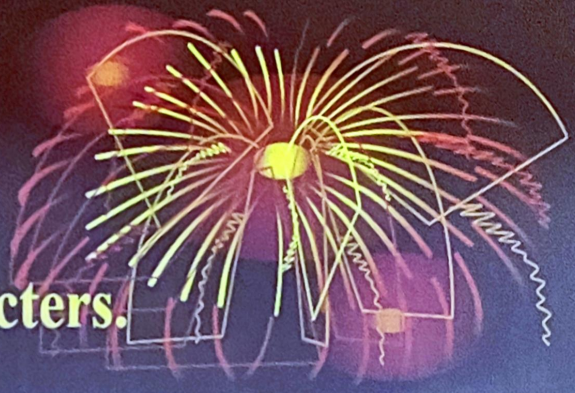
coontry[0]	E
coontry[1]	s
coontry[2]	w
coontry[3]	a
coontry[4]	t
coontry [5]	i
coontry[6]	n
coontry[7]	i
coontry[8]	\n

Wrong	Comments
`int 3a, 1, -p;`	Starts with digit or -
`short number+one;`	Has a +
`long #number;`	Starts with #

Which of these identifiers are correct?

- AREA
  area_under_the_curve
- 3D num45
- Last-Chance #values
- x_yt3 pi
- num$ %done
- lucky***

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 reflect its inherent meaning to make the program comprehensible (*Good programming style*)
- E.g. *length, base, perimeter_of_rectangle* clearly specify what the identifier represents

Constants

- A data item whose value does not change
- Constants can have names and follow the same rules as that of variable names
- Simple numbers like 123, 0.24 are of course constants (which are without name)
- E.g. a = 25; // 25 is the constant

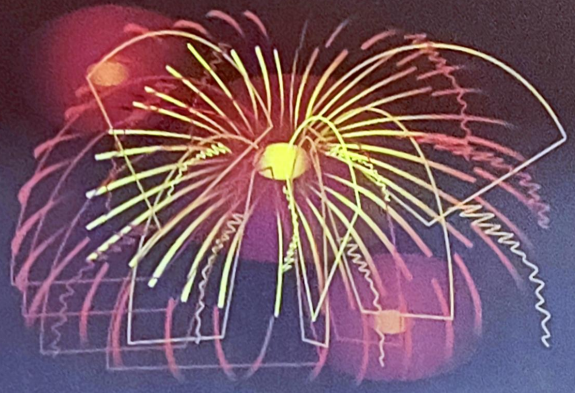
Constants ... (Named constants)

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

- e.g.: *const int* max_number = 100;

  const float pi = 3.14159;

The #define preprocessor directive or Macro



- An easier way to save typing/writing in a program when a variable name or constant is referred to 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.ac.in"
- 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 27

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 27 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.

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)

Declarations

- An identifier which you want to use should be *declared* first.
- A declaration specifies *type* and *name* of the identifiers.
- Optionally a declaration for a variable can contain its initial value also.

Example:

```
int value = 55;
```

Declarations ...

- int count; /* declares that *count* is a variable which can hold integer values */
- char s, c; /* s and c are variables which can hold a single character each */
- float sum_1 = 0.5;
- double d, total, sum;
- Declaration is a *statement* and every *statement* in C should end with a semicolon ;

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.

int

- ints are generally stored as 4 bytes (system dependent)
- For 4 bytes:

  $-2^{31}, -2^{31} + 1, \ldots, -3, -2, -1, 0, 1, 2, 3, \ldots, 2^{31}-1$

  Min Int = -2^{31}

  Max Int = $2^{31} - 1$

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.

Declarations ...

- Size of *char* is 1 byte (8 bits) - there can be only 2^8 different characters that you can use.
- Sizes

  char → 1 byte

  int → 4 bytes

  float → normally 4 bytes

  double → double the size of float

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);
}
```

Testing size of Numeric Data

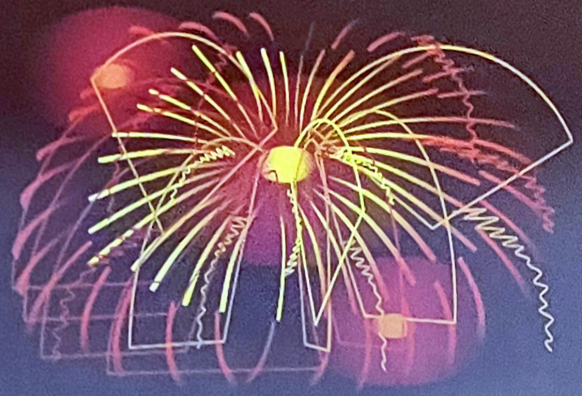
```
#include<stdio.h>
int main(){
  printf("size of char %d\n", sizeof(char)); //1
  printf("size of short %d\n",sizeof(short)); //2
  printf("size of int %d\n",sizeof(int)); //4
  printf("size of long int %d\n",sizeof(long int)); //8

  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;
}
```

Constants ...

- 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 single quotes, e.g.

  'x'

- Actually a character is stored as an integer.
- The integer value of a character is often called as its ASCII value.
  (*In 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
27	033	1B	00011011	ESC	&#027;		Escape

Constants

- Some characters can not 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;
- There are more escape characters which you should read about them from the book as an exercise.