

# CSC 01 - Introduction to Computing

**Dr. Saravanan Chandran, Ph.D. (NIT-T)**  
Associate Professor, Dept. of Computer Science and Engineering,  
NIT Durgapur [cs@cse.nitdgp.ac.in](mailto:cs@cse.nitdgp.ac.in)  
Mob. +91-94347-88036  
<https://sites.google.com/view/drcs>

Prog. in C - Dr. Chandran Saravanan, NITDGP

## References

- institute library, internet
- <https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html>
- [https://www-s.acm.illinois.edu/webmonkeys/book/c\\_guide/](https://www-s.acm.illinois.edu/webmonkeys/book/c_guide/)
- <https://www.codechef.com/ide>
- <https://turboc.codeplex.com>

Prog. in C - Dr. Chandran Saravanan, NITDGP



Prog. in C - Dr. Chandran Saravanan, NITDGP

## COMPUTER - COMPUTING

- An electronic device which is capable of receiving information (data) in a particular form and of performing a sequence of operations in accordance with a predetermined but variable set of procedural instructions (program) to produce a result in the form of information or signals.
- the use or operation of computers
- Usage of a computers
  - Word Processing, Web Surfing, Instant Messaging, Email, Music, Movies, Games
  - Air traffic control, Car diagnostics, Climate control, etc.

Prog. in C - Dr. Chandran Saravanan, NITDGP

## Computer Operations

An electronic device

- receives information (data)
- performs a sequence of operations (calculations)
- as per the set of procedural instructions (programs)
- produces result (reports)

Prog. In C - Dr. Chandran Saravanan, NITDGP



Prog. In C - Dr. Chandran Saravanan, NITDGP

## Types of Computers

- **Microcomputers (personal computers)** single chip [microprocessors](#)
- [Desktop computers](#) - A case and a display put on a desk.
- [Laptops](#) and [notebook computers](#) – Portable and all in two foldable cases.
- [Tablet computer](#) – Portable and all in one case.
- [Smartphones](#) – Small handheld computers with limited hardware.

Prog. In C - Dr. Chandran Saravanan, NITDGP



Prog. In C - Dr. Chandran Saravanan, NITDGP

## Servers

- usually refers to a computer that is dedicated to provide a service
- For example, a computer dedicated to a [database](#) may be called a "[database server](#)".
- "[File servers](#)" manage a large collection of [computer files](#).
- "[Web servers](#)" process [web pages](#) and [web applications](#).

Prog. in C - Dr. Chandran Saravanan, NITDGP



Prog. in C - Dr. Chandran Saravanan, NITDGP

## Workstations

- A **workstation** is a special **computer** designed for technical or scientific applications.
- Primarily used by one person at a time.
- They are connected to a local area network and run multi-user operating systems.

Prog. in C - Dr. Chandran Saravanan, NITDGP



Prog. in C - Dr. Chandran Saravanan, NITDGP

## Minicomputers

- [Minicomputers](#) are a class of multi-user [computers](#) that lie in the middle range of the computing spectrum, in between the smallest [mainframe computers](#) and the largest single-user systems ([microcomputers](#) or [personal computers](#)).

Prog. in C - Dr. Chandran Saravanan, NITDGP



Prog. in C - Dr. Chandran Saravanan, NITDGP

## Mainframe

- The term [mainframe computer](#) was created to distinguish the traditional, large, institutional computer intended to service multiple users from the smaller, single user machines. They are measured in [MIPS](#) (million instructions per second) and respond to up to 100s of millions of users at a time.

Prog. in C - Dr. Chandran Saravanan, NITDGP

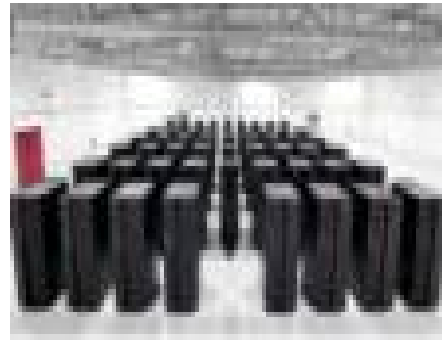


Prog. in C - Dr. Chandran Saravanan, NITDGP

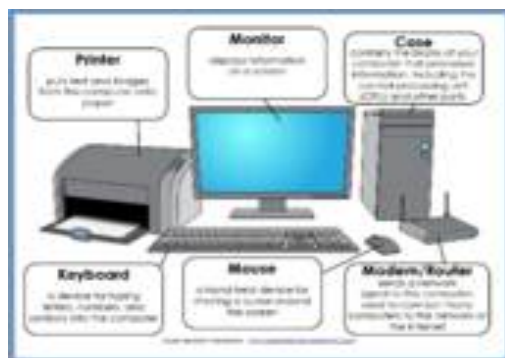
## Supercomputers

- A Supercomputer is focused on performing tasks involving intense numerical calculations such as weather forecasting, fluid dynamics, nuclear simulations, theoretical astrophysics, and complex scientific computations. Supercomputer processing speeds are measured in floating point operations per second, or [FLOPS](#).

Prog. in C - Dr. Chandran Saravanan, NITDGP



Prog. in C - Dr. Chandran Saravanan, NITDGP



Prog. in C - Dr. Chandran Saravanan, NITDGP



Prog. in C - Dr. Chandran Saravanan, NITDGP



## Cabinet-Mother Board-Processors

- Cabinet: SMPS 450, 500, 550, 600, 650, 750, 850, 900 WATT
- Mother Board: Intel, Zebronics, Asus, Gigabyte, Mercury, Frontech, Msi, Ecs, Chipset, Memoryslots, Form Factor (size, configuration)
- Processors: AMD A10-5800K, AMD FX-9590, AMD Sempron 3850, Intel Core i3-6100, Intel Core i7-6700K, Intel Core i5-4690K, CPU Cores 4, 8, GPU Cores, CPU Frequency, 3M, 6M, 8M, Cache (L2), 3.00GHz, 3.1GHz, 3.2GHz, 3.4GHz, 3.5GHz, 3.9GHz, 4.0GHz, 4.2GHz, 4.40GHz, first, second, third, fourth generation.

Prog. In C - Dr Chandran Saravanan, NITDGP

## Memory-Storage

- ROM / RAM: BIOS, DDR1, DDR2, DDR3, DDR4, 2GB, 4GB, 8GB.
- HDD: HP, Kingston, ADATA, Sony, Toshiba, Seagate, WD, Barracuda – SSD, SATA, SCSI, 5400RPM, 7200RPM, 10,000RPM, 500GB, 1TB, 2TB, 3TB.
- BD, DVD, CD: HP, LG, Sony, Samsung. 8x, 18x, 24x, 48x, SATA, RW, Cache, Form Factor 5.25".
- USB – 2.0, 3.0, 3.1, Keyboard – QWERTY, Mouse – Optical mouse, left, right click, scroll

Prog. In C - Dr Chandran Saravanan, NITDGP

## Monitors

- **CRT (Cathode Ray Tube)**, TFT, LCD (liquid crystal display), LED (Light Emitting Diode), OLED (Organic LED), Plasma.
- **Screen Size** (13, 15, 17, 19, 21, 23, 27 and 32 inches),
- **Viewing Angle** (The viewing angle indicates at what angle the monitor can be viewed vertically and horizontally and still be seen.),
- **Contrast Ratio** (The contrast ratio determines how rich colors will appear on-screen, the higher the ratio the better. Contrast ratios range from 200:1 up to 1000:1),
- **Resolution and Refresh Rates** The resolution is the number of dots displayed on the entire screen. The higher the resolution the smaller everything on the screen will be. This can be a benefit for running multiple applications at the same time but can also be a burden for someone with poor eyesight. Common resolution supports include 640 \* 480, 800 \* 600 and 1,024\*768 and so forth. The refresh rate of a monitor is the frequency at which the screen is redrawn. The higher the number the more often the screen is redrawn and the less flicker will occur. Common Refresh rate are 60 to 80Hertz.

Prog. in C - Dr Chandran Saravanan, NITDGP

## Printers-Speakers

- **Printers:** Dot matrix (80/132 column), Inkjet, Laserjet, A8(Business card-2.07"x2.91") /A7...A4(8.27"x11.69")/A3/A0/2A Plotter, Black & White, Colour.
- **Speakers** – Mono, Stereo, Home Theatre, 2.1, 5.1, 7.1, SubWoofer, Wireless, Bluetooth.

Prog. in C - Dr Chandran Saravanan, NITDGP

## Software

- **System Software:** Dedicated to managing the computer itself, such as the operating system, file management utilities, and disk operating system. The computer programs used to start and run computer systems.
- **Application Software:** Specific purpose programs word processing, web browsers, accounting, truck scheduling, Astrology, Music Player, Movie Player, Video Games, etc.

Prog. in C - Dr Chandran Saravanan, NITDGP

## Programming Languages

- **Language (High / Low Level)** is set of instructions to perform specific task.
- **High level languages** use common simple English words for instructions. No need of detailed information about computer hardware. Example: C, C++, Java, etc.
- **Low level languages** use specific symbols for instructions. Detailed information is required about computer hardware. Example: Assembly Language.

Prog. in C - Dr Chandran Saravanan, NITDGP

## Compiler-Interpreter

- **Compiler:** is a computer program (or a set of programs) that transforms source code (Program) written in a programming language (the source language) into another computer language / object code (binary language).
- **Interpreter:** is a computer program that directly executes, i.e. performs, instructions written in a programming or scripting language, without previously compiling them into a machine language program.

Prog. in C - Dr. Chandran Saravanan, NITDGP

## Booting

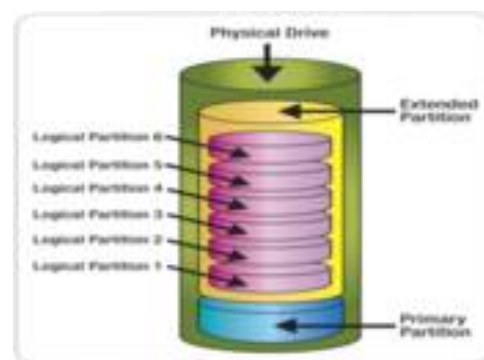
- is the initialization of a computerized system.
- The booting process can be "hard", after electrical power to the CPU is switched from off to on (in order to diagnose particular hardware errors),
- or "soft", when those power-on self-tests (POST) can be avoided.
- A boot loader is a computer program that loads an operating system or some other system software for the computer after completion of the power-on self-tests;
- it is the loader for the operating system itself. Within the hard reboot process, it runs after completion of the self-tests, then loads and runs the software.
- A boot loader is loaded into main memory from persistent memory, such as a hard disk drive.

Prog. in C - Dr. Chandran Saravanan, NITDGP

## Disk Partitions- File System

- Partitioning is typically the first step of preparing a newly manufactured disk, before any files or directories have been created.
- The disk stores the information about the partitions' locations and sizes in an area known as the partition table that the operating system reads before any other part of the disk.
- Each partition then appears in the operating system as a distinct "logical" disk that uses part of the actual disk. System administrators use a program called a partition editor to create, resize, delete, and manipulate the partitions.
- In windows, Go to Control Panel, Disk Management for managing partitions.

Prog. in C - Dr. Chandran Saravanan, NITDGP



Prog. in C - Dr. Chandran Saravanan, NITDGP



## Multimedia

- is content that uses a combination of different content forms such as text, audio, images, animation, video and interactive content.
- Graphics are visual images or designs on some surface, such as a wall, canvas, screen, paper, or stone to inform, illustrate, or entertain.
- In contemporary usage it includes: pictorial representation of data, as in computer-aided design and manufacture, in typesetting and the graphic arts, and in educational and recreational software.
- Images that are generated by a computer are called computer graphics. Applications: cinema presentation, video game, simulator, etc.
- Animation is the process of making the illusion of motion and change by means of the rapid display of a sequence of static images that minimally differ from each other. Images are displayed in a rapid succession, usually 24, 25, 30, or 60 frames per second.

Prog. in C - Dr. Chandran Saravanan, NITDGP

## Number System

- Binary number system – 0 and 1 – base 2
- Decimal number system – 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 – base 10
- Octal number system – 0, 1, 2, 3, 4, 5, 6, 7 – base 8
- Hexa Decimal number system – 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F – base 16

Prog. in C - Dr. Chandran Saravanan, NITDGP

## Decimal to Binary Conversion

- **Step 1** – Divide the decimal number to be converted by 2 (the value of the new base).
- **Step 2** – Get the remainder from Step 1 as the rightmost digit (least significant digit) of new base number.
- **Step 3** – Divide the quotient of the previous divide by 2 (the new base).
- **Step 4** – Record the remainder from Step 3 as the next digit (to the left) of the new base number.

Prog. in C - Dr. Chandran Saravanan, NITDGP

## Example

Steps	Operations	Results	Remainder
Step 1	29 / 2	14	1
Step 2	14 / 2	7	0
Step 3	7 / 2	3	1
Step 4	3 / 2	1	1
Step 5	1 / 2	0	1

Prog. in C - Dr. Chandran Saravanan, NITDGP

### Example continues ...

- As mentioned in Steps 2 and 4, the remainders have to be arranged in the reverse order so that the first remainder becomes the Least Significant Digit (LSD) and the last remainder becomes the Most Significant Digit (MSD).
- Decimal Number –  $29_{10}$  = Binary Number –  $11101_2$ .

Prog. in C - Dr Chandran Saravanan, NITDGP

### Find the Binary equivalent for Decimal 35

2	35	1	<div>↑ LSD</div>
2	18	0	
2	9	1	
2	4	0	
2	2	0	
2	1	1	
	0		
Divisor		Quotient	Remainder

Quotient

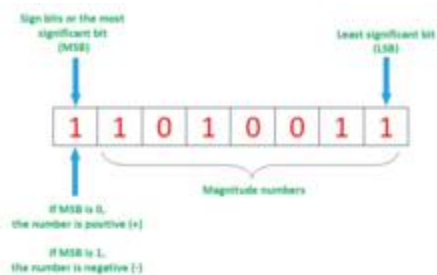
MSD - most significant digit

LSD - least significant digit

Therefore, the binary equivalent for 35 is

**100101**

Prog. in C - Dr Chandran Saravanan, NITDGP



Prog. in C - Dr Chandran Saravanan, NITDGP

### Binary to Decimal Conversion

- Step 1** – Determine the column (positional) value of each digit (this depends on the position of the digit and the base of the number system).
- Step 2** – Multiply the obtained column values (in Step 1) by the digits in the corresponding columns.
- Step 3** – Sum the products calculated in Step 2. The total is the equivalent value in decimal.

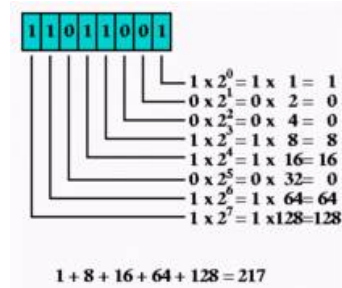
Prog. in C - Dr Chandran Saravanan, NITDGP

## Example

Steps	Binary Number	Decimal Number
Step 1	10101 <sub>2</sub>	$((1 \times 24) + (1 \times 23) + (1 \times 22) + (0 \times 21) + (1 \times 20))10$
Step 2	10101 <sub>2</sub>	$(16 + 8 + 4 + 0 + 1)10$
Step 3	10101 <sub>2</sub>	2910

Binary Number – 11101<sub>2</sub> = Decimal Number – 29<sub>10</sub>

Prog. In C - Dr Chandran Saravanan, NITDGP



Prog. In C - Dr Chandran Saravanan, NITDGP

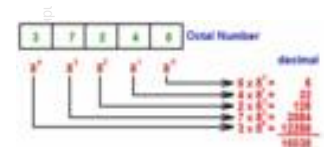
## Octal Number System

- Characteristics of the octal number system are as follows – Uses eight digits, 0, 1, 2, 3, 4, 5, 6, 7
- Also called as base 8 number system
- Each position in an octal number represents a 0 power of the base (8). Example 8<sup>0</sup>
- Last position in an octal number represents a x power of the base (8).
- Example 8<sup>x</sup> where x represents the last position - 1
- Example Octal Number: 12570<sub>8</sub>

Prog. In C - Dr Chandran Saravanan, NITDGP

## Octal to Decimal Conversion

Steps	Octal Number	Decimal Number
Step 1	12570 <sub>8</sub>	$((1 \times 8^4) + (2 \times 8^3) + (5 \times 8^2) + (7 \times 8^1) + (0 \times 8^0))_{10}$
Step 2	12570 <sub>8</sub>	$(4096 + 1024 + 320 + 56 + 0)_{10}$
Step 3	12570 <sub>8</sub>	5496 <sub>10</sub>



Prog. In C - Dr Chandran Saravanan, NITDGP

## Hexadecimal Number System

- Characteristics of hexadecimal number system are as follows – Uses 10 digits and 6 letters, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
- Letters represent the numbers starting from 10. A = 10, B = 11, C = 12, D = 13, E = 14, F = 15
- Also called as base 16 number system
- Each position in a hexadecimal number represents a 0 power of the base (16). Example, 160
- Last position in a hexadecimal number represents a x power of the base (16). Example 16x where x represents the last position - 1
- Example Hexadecimal Number: 19FDE<sub>16</sub>

Prog. In C - Dr Chandran Saravanan, NITDGP

## Hexadecimal to Decimal Conversion

Steps	Hexadecimal Number	Decimal Number
Step 1	19FDE <sub>16</sub>	$((1 \times 16^4) + (9 \times 16^3) + (F \times 16^2) + (D \times 16^1) + (E \times 16^0))_{10}$
Step 2	19FDE <sub>16</sub>	$((1 \times 16^4) + (9 \times 16^3) + (15 \times 16^2) + (13 \times 16^1) + (14 \times 16^0))_{10}$
Step 3	19FDE <sub>16</sub>	$(65536 + 36864 + 3840 + 208 + 14)_{10}$
Step 4	19FDE <sub>16</sub>	106462 <sub>10</sub>

Prog. In C - Dr Chandran Saravanan, NITDGP

## Hexadecimal to Decimal Conversion

Convert 3B4F to its decimal equivalent:

Hex Digits	→	3	B	4	F
Positional Values	→	$16^3$	$16^2$	$16^1$	$16^0$
Products	→	$12288$	$+2816$	$+64$	$+15$
		<b>15,183<sub>10</sub></b>			

Prog. In C - Dr Chandran Saravanan, NITDGP

## Decimal to Other Base System

- Step 1 – Divide the decimal number to be converted by the value of the new base.
- Step 2 – Get the remainder from Step 1 as the rightmost digit (least significant digit) of the new base number.
- Step 3 – Divide the quotient of the previous divide by the new base.
- Step 4 – Record the remainder from Step 3 as the next digit (to the left) of the new base number.
- Repeat Steps 3 and 4, getting remainders from right to left, until the quotient becomes zero in Step 3.
- The last remainder thus obtained will be the Most Significant Digit (MSD) of the new base number.
- Example Decimal Number: 29<sub>10</sub>

Prog. In C - Dr Chandran Saravanan, NITDGP

Decimal Base 10	Binary Base 2	Octal Base 8	Hexa Decimal Base 16
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10

Prog. In C - Dr. Chandran Saravanan, NITDGP

## Decimal to Binary

Steps	Operation	Result	Remainder
Step 1	29 / 2	14	1
Step 2	14 / 2	7	0
Step 3	7 / 2	3	1
Step 4	3 / 2	1	1
Step 5	1 / 2	0	1

Prog. In C - Dr. Saravanan Chandran, NITDGP

## Other Base System to Non-Decimal System

- Step 1 – Convert the original number to a decimal number (base 10).
- Step 2 – Convert the decimal number so obtained to the new base number.

Prog. In C - Dr. Chandran Saravanan, NITDGP

## Shortcut Method – Binary to Octal

- Step 1 – Divide the binary digits into groups of three (starting from the right).
- Step 2 – Convert each group of three binary digits to one octal digit.

Steps	Binary Number	Octal Number
Step 1	10101 <sub>2</sub>	010 101
Step 2	10101 <sub>2</sub>	2 <sub>8</sub> 5 <sub>8</sub>
Step 3	10101 <sub>2</sub>	25 <sub>8</sub>

Prog. In C - Dr. Chandran Saravanan, NITDGP

### Shortcut Method – Octal to Binary

- Step 1 – Convert each octal digit to a 3-digit binary number (the octal digits may be treated as decimal for this conversion).
- Step 2 – Combine all the resulting binary groups (of 3 digits each) into a single binary number.

Steps	Octal Number	Binary Number
Step 1	25 <sub>8</sub>	2 <sub>10</sub> 5 <sub>10</sub>
Step 2	25 <sub>8</sub>	010 <sub>2</sub> 101 <sub>2</sub>
Step 3	25 <sub>8</sub>	010101 <sub>2</sub>

Prog. in C - Dr Chandran Saravanan, NITDGP

### Shortcut Method – Binary to Hexadecimal

- Step 1 – Divide the binary digits into groups of four (starting from the right).
- Step 2 – Convert each group of four binary digits to one hexadecimal symbol.

Steps	Binary Number	Hexadecimal Number
Step 1	10101 <sub>2</sub>	0001 0101
Step 2	10101 <sub>2</sub>	1 <sub>16</sub> 5 <sub>16</sub>
Step 3	10101 <sub>2</sub>	15 <sub>16</sub>

Prog. in C - Dr Chandran Saravanan, NITDGP

### Shortcut Method - Hexadecimal to Binary

- Step 1 – Convert each hexadecimal digit to a 4-digit binary number (the hexadecimal digits may be treated as decimal for this conversion).
- Step 2 – Combine all the resulting binary groups (of 4 digits each) into a single binary number.

Steps	Hexadecimal Number	Binary Number
Step 1	15 <sub>16</sub>	1 <sub>10</sub> 5 <sub>10</sub>
Step 2	15 <sub>16</sub>	0001 <sub>2</sub> 0101 <sub>2</sub>
Step 3	15 <sub>16</sub>	00010101 <sub>2</sub>

Prog. in C - Dr Chandran Saravanan, NITDGP

### Decimal fraction to Binary

1. Multiply the fractional decimal number by 2
  2. Integral part of resultant decimal number will be first digit of fraction binary number
  3. Repeat step 1 using only fractional part of decimal number and then step 2
- $0.47_{10} * 2_{10} = 0.94_{10}$ , Integral part: 0<sub>2</sub>
  - $0.94_{10} * 2_{10} = 1.88_{10}$ , Integral part: 1<sub>2</sub>
  - $0.88_{10} * 2_{10} = 1.76_{10}$ , Integral part: 1<sub>2</sub>
  - $0.47_{10} = 0.011_2$

Prog. in C - Dr Saravanan Chandran, NITDGP

## Binary fraction to Decimal

1. Divide each digit from right side of radix point till the end by  $2^1, 2^2, 2^3, \dots$  respectively.
  2. Add all the result coming from step 1.
- Equivalent fractional decimal number would be the result obtained in step 2.
  - $\Rightarrow 0.101_2 = (1 \cdot 1/2) + (0 \cdot 1/2^2) + (1 \cdot 1/2^3)$
  - $\Rightarrow 0.101_2 = 1 \cdot 0.5 + 0 \cdot 0.25 + 1 \cdot 0.125$
  - $\Rightarrow 0.101_2 = 0.625_{10}$

Prog. in C - Dr. Saravanan Chandran, NITDGP

## Decimal fraction to Hexa

- multiply 0.00390625 by 16 and take the integer part
- $0.00390625 \times 16 = 0.0625$ ,
- Integer part = 0,
- Fractional part = 0.0625
- multiply 0.0625 by 16 and take the integer part
- $0.0625 \times 16 = 1.000$
- Integer part = 1
- Fractional part = 0
- $0.00390625_{10} = 0.01_{16}$

Prog. in C - Dr. Saravanan Chandran, NITDGP

## Hexa fraction to Decimal

- $0.16_{16}$
- multiply 1 by  $1/16$  and take the integer part
- $1 \times 1/16 = 0.0625$
- multiply 6 by 16 and take the integer part
- $6 \times 1/16^2 = 0.023$
- $0.16_{16} = 0.648_{10}$

Prog. in C - Dr. Saravanan Chandran, NITDGP

## Decimal fraction to Octal

- multiply 0.015625 by 8 and take the integer part
- $0.015625 \times 8 = 0.125$
- Integer part = 0
- Fractional part = 0.125
- multiply 0.125 by 8 and take the integer part
- $0.125 \times 8 = 1.000$
- Integer part = 1
- Fractional part = 0
- $0.015625_{10} = 0.01_8$

Prog. in C - Dr. Saravanan Chandran, NITDGP

## Octal fraction to Decimal

- $0.25_8$
  - multiply 1 by  $1/8$  and take the integer part
  - $2 \times 1/8 = 0.25_{10}$  integral part = 0
  - multiply 6 by  $1/8^2$  and take the integer part
  - $5 \times 1/8^2 = 0.078125_{10}$
- $0.25_8 = 0.328125_{10}$

Prog. in C - Dr. Saravanan Chandran, NITDGP

## ASCII

ASCII stands for American Standard Code for Information Interchange. Computers can only understand numbers, so an ASCII code is the numerical representation of a character such as 'a' or '@' or an action of some sort. 0-31 Control keys, 32-47 special characters, 48-57 numbers, 58-64 special characters, 65-90 capital letters, 91-96 special characters, 97-122 small letter, 123-127 special characters, 128-255 extended ASCII codes. If a key is pressed respective ASCII code is identified and converted to binary.

Prog. in C - Dr. Chandran Saravanan, NITDGP

## Unicode

- Unicode is a computing industry standard for the consistent encoding, representation, and handling of text expressed in most of the world's writing systems.
- Character Decimal Hex Name

অ      2437 0985      BENGALI LETTER      A

Prog. in C - Dr. Chandran Saravanan, NITDGP

## Algorithm

- In computer science, an algorithm is a
  - finite sequence of well-defined,
  - computer-implementable instructions,
  - typically to solve a class of problems or
  - to perform a computation.
- Algorithms are always unambiguous and are used as specifications for performing calculations, data processing, automated reasoning, and other tasks.

Prog. in C - Dr. Saravanan Chandran, NITDGP



## Example

Algorithm LargestNumber

Input: A list of numbers L.

Output: The largest number in the list L.

```

if L.size = 0 return null
largest ← L[0]
for each item in L, do
  if item > largest, then
    largest ← item
return largest

```

Prog. In C - Dr. Saravanan Chandran, NITDGP

## Flowchart

- A flowchart is a graphical representations of steps. It was originated from computer science as a tool for representing algorithms and programming logic but had extended to use in all other kinds of processes.
- Nowadays, flowcharts play an extremely important role in displaying information and assisting reasoning.
- They help us visualize complex processes, or make explicit the structure of problems and tasks.
- A flowchart can also be used to define a process or project to be implemented.

Prog. In C - Dr. Saravanan Chandran, NITDGP

### Terminator

The terminator symbol represents the starting or ending point of the system.



### Process

A box indicates some particular operation.



Prog. In C - Dr. Saravanan Chandran, NITDGP

### Decision

A diamond represents a decision or branching point. Lines coming out from the diamond indicates different possible situations, leading to different sub-processes.

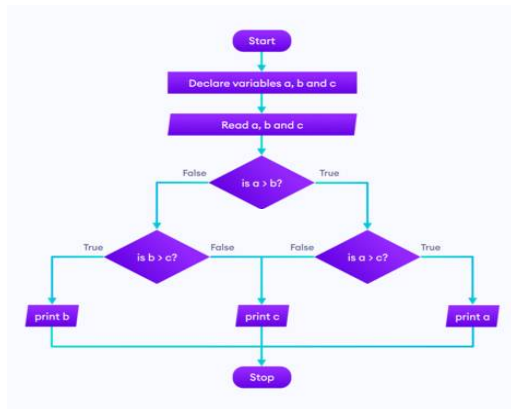


### Data

It represents information entering or leaving the system. An input might be an order from a customer. Output can be a product to be delivered.



Prog. In C - Dr. Saravanan Chandran, NITDGP



Prog. In C - Dr. Saravanan Chandran, NITDGP

## Programming Language

- Communication tool
- Highest - Pascal, COBOL, FORTRAN, BASIC
- Middle - Java, C++, C
- Lowest - Assembler

Prog. In C - Dr. Chandran Saravanan, NITDGP

## Origins of C

- Invented and implemented by Dennis Ritchie
- 1970
- DEC PDP 11
- Unix operating system
- developed from BCPL by Martin Richards
- influence of B by Ken Thompson
- standardised by Brian Kernighan and D Ritchie

Prog. In C - Dr. Chandran Saravanan, NITDGP

## C Fundamentals

- The basic elements used to construct a C program are:
  - the C character set,
  - identifiers and keywords,
  - data types,
  - constants,
  - arrays,
  - declarations,
  - expressions and
  - statements.

Prog. In C - Dr. Chandran Saravanan, NITDGP

## The C Character Set

- C uses letters A to Z in lowercase and uppercase, the digits 0 to 9, certain special characters, and white spaces to form basic program elements (e.g variables, constants, expressions etc.) The special characters are:
- + - \* / = % & # ! ? ^ " ' / | < > ( ) [ ] { } : ; , ~ @ !
- The white spaces used in C programs are: blank space, horizontal tab, carriage return, new line and form feed.

Prog. in C - Dr Chandran Saravanan, NITDGP

## Identifiers and Keywords

- Identifiers are names given to various program elements such as variables, functions, and arrays.
- Identifiers consist of letters and digits, in any order, except that the first character must be a letter.
- Both uppercase and lowercase letters are permitted and the underscore may also be used, as it is also regarded as a letter.
- Uppercase and lowercase letters are not equivalent, C is case sensitive. An identifier can be arbitrarily long.
- The same identifier may denote different entities in the same program, for example, a variable and an array may be denoted by the same identifier, example below.
- `int sum, average, A[10]; // sum, average and the array name A are all identifiers.`

Prog. in C - Dr Chandran Saravanan, NITDGP

## Identifiers and Keywords cont...

- **The `__func__` predefined identifier:-**
- The predefined identifier `__func__` makes a function name available for use within the function.
- Immediately following the opening brace of each function definition, `__func__` is implicitly declared by the compiler in the following way:
- `static const char __func__[ ] = "function-name";` where function-name is the name of the function.
- Example
- `#include <stdio.h> void func1(void) { printf("%sn", __func__); return;} int main() { func1();}`
- The output would be `func1`

Prog. in C - Dr Chandran Saravanan, NITDGP

## Keywords

- Keywords are reserved words that have standard predefined meanings.
- These keywords can only be used for their intended purpose;
- they cannot be used as programmer defined identifiers.
- There are **32 Keywords** in C
- Examples of some keywords are: `int`, `main`, `void`, `if`.

Prog. in C - Dr Chandran Saravanan, NITDGP

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

Prog. In C - Dr Chandran Saravanan, NITDGP

## Data Types

- char 8bit -127 to 127
- unsigned char 8bit 0 to 255
- int 16 or 32bit -32,768 to 32,767
- unsigned int 16 or 32bit 0 to 65,535
- long int 32bit -2,147,483,648 to 2,147,483,647
- unsigned long int 32bit 0 to 4,294,967,295
- float 32bit 6 decimal places
- double 64bit 15 decimal places
- long double 80bit 19 decimal places

Prog. In C - Dr Chandran Saravanan, NITDGP

## int

- It is used to store an integer quantity. An ordinary int can store a range of values from INT\_MIN to INT\_MAX as defined by in header file <limits.h>.
- The type modifiers for the int data type are: signed, unsigned, short, long and long long.
- A short int occupies 2 bytes of space and a long int occupies 4 bytes.
- A short unsigned int occupies 2 bytes of space but it can store only positive values in the range of 0 to 65535.
- An unsigned int has the same memory requirements as a short unsigned int. However, in case of an ordinary int, the leftmost bit is reserved for the sign.
- A long unsigned int occupies 4 bytes of memory and stores positive integers in the range of 0 to 4294967295.
- By default the int data type is signed.
- A long long int occupies 64 bits of memory. It may be signed or unsigned. The signed long long int stores values from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 and
- the unsigned long long ranges from 0 to 18,446,744,073,709,551,615.

Prog. In C - Dr Chandran Saravanan, NITDGP

## char

- It stores a single character of data belonging to the C character set.
- It occupies 1 byte of memory, and stores any value from the C character set.
- The type modifiers for char are signed and unsigned.
- Both signed and unsigned char occupy 1 byte of memory but the range of values differs.
- An unsigned char can store values from 0 to 255 and a signed char can store values from -128 to +127.
- Each char type has an equivalent integer interpretation, so that a char is really a special kind of short integer.
- By default, char is unsigned.

Prog. In C - Dr Chandran Saravanan, NITDGP

## float and double

- float is used to store real numbers with single precision i.e. a precision of 6 digits after decimal point. It occupies 4 bytes of memory.
- The type modifier for float is long. It has the same memory requirements as double.
- double is used to store real numbers with double precision. It occupies 8 bytes of memory.
- The type modifier for double is long. A long double occupies 10 bytes of memory.

Prog. in C - Dr Chandran Saravanan, NITDGP

## void, \_Bool, \_Complex

- void is used to specify an empty set containing no values. Hence, it occupies 0 bytes of memory.
- A boolean data type, which is an unsigned integer type, that can store only two values, 0 and 1.
- It is defined in <stdbool.h>
- \_Complex is used to store complex numbers. There are three complex types: float \_Complex, double \_Complex, and long double \_Complex
- It is defined in the <complex.h> file.

Prog. in C - Dr Chandran Saravanan, NITDGP

## Variables

- Location of a memory
- Holds a value
- Assigned a name

### Examples

```
int    apple, mango;
char   student, faculty;
float  amount, tax;
```

Prog. in C - Dr Chandran Saravanan, NITDGP

## Local or Global variables

- Local  
Variables declared inside a function  
not available to other functions
- Global  
Variables declared outside a function  
available to other functions

Prog. in C - Dr Chandran Saravanan, NITDGP

## Constant and Volatile

- **const**  
value will not change throughout the program  
initial value remains same
- **volatile**  
value will be set or changed by environment  
real time systems / embedded programs

Prog. in C - Dr Chandran Saravanan, NITDGP

## Constants

- A constant is an identifier whose value remains unchanged throughout the program.
- To declare any constant the syntax is:
- `const datatype varname = value;`
- where `const` is a keyword that declares the variable to be a fixed value entity.
- There are four basic types of constants in C. They are integer constants, floating point constants, character constants and string constants.
- Integer and floating point constants cannot contain commas or blank spaces; but they can be prefixed by a minus sign to indicate a negative quantity.

Prog. in C - Dr Chandran Saravanan, NITDGP

## Integer Constants

- An integer constant is an integer valued number. It consists of a sequence of digits. Integer constants can be written in the following three number systems:
- **Decimal (base 10):** A decimal constant can consist of any combination of digits from 0 to 9. If it contains two or more digits, the first digit must be something other than 0, for example: `const int size = 50;`
- **Octal (base 8):** An octal constant can consist of any combination of digits from 0 to 7. The first digit must be a 0 to identify the constant as an octal number, for example: `const int a = 074;` `const int b = 0;`
- **Hexadecimal constant (base 16):** A hexadecimal constant can consist of any combination of digits from 0 to 9 and a to f (either uppercase or lowercase). It must begin with 0x or 0X to identify the constant as a hexadecimal number, for example: `const int c = 0x7FF;`
- Integer constants can also be prefixed by the type modifiers `unsigned` and `long`. Unsigned constants must end with `u` or `U`, long integer constants must end with `l` or `L` and unsigned long integer constants must end with `ul` or `UL`. Long long integer constants end with `LL` or `ll`. Unsigned long long end with `ULL` or `ull`.

Prog. in C - Dr Chandran Saravanan, NITDGP

## Floating Point Constant

- Its a base 10 or a base 16 number that contains a decimal point or an exponent or both.
- In case of a decimal floating point constant the exponent the base 10 is replaced by `e` or `E`. Thus,  $1.4 \times 10^{-3}$  would be written as `1.4E-3` or `1.4e-3`.
- In case of a hexadecimal character constant, the exponent is in binary and is replaced by `p` or `P`.
- For example:
- `const float a = 5000.;` `const float b = .1212e12;` `const float c = 827.54;`
- Floating point constants are generally double precision quantities that occupy 8 bytes. In some versions of C, the constant is appended by `F` to indicate single precision and by `L` to indicate a long floating point constant.

Prog. in C - Dr Chandran Saravanan, NITDGP

## Character Constants

- A character constant is a sequence of one or more characters enclosed in apostrophes. Each character constant has an equivalent integer value that is determined by the computer's character set. It may also contain escape sequences. A character literal may be prefixed with the letter L, u or U, for example 'L'c'.
- A character literal without the L prefix is an ordinary character constant or a narrow character constant.
- A character literal with the L prefix is a wide character constant.
- The type of a narrow character constant and a multicharacter constant is int.
- The type of a wide character constant with prefix L is wchar\_t defined in the header file <stdint.h>.
- A wide character constant with prefix u or U is of type char16\_t or char32\_t. These are unsigned character types defined in <uchar.h>.
- An ordinary character literal that contains more than one character or escape sequence is a multicharacter constant, for example: const char p= 'A';
- Escape Sequences are also character constants that are used to express certain non printing characters such as the tab or the carriage return.
- An escape sequence always begins with a backward slash and is followed by one or more special characters. for eg. b will represent the bell, n will represent the line feed.

Prog. in C - Dr Chandran Saravanan, NITDGP

## String Literals

- A string literal consists of a sequence of multibyte characters enclosed in double quotation marks.
- They are of two types, wide string literal and UTF-8 string literal. A UTF-8 string literal is prefixed by u8 and a wide string literal by L, u, or U.
- The compiler recognizes and supports the additional characters (the extended character set) which you can meaningfully use in string literals and character constants.
- The support for extended characters includes the multibyte character sets. A multibyte character is a character whose bit representation fits into one or more bytes.

Prog. in C - Dr Chandran Saravanan, NITDGP

## Symbolic Constants

- A symbolic constant is a name that substitutes for a numeric constant, a character constant or a string constant throughout the program. When the program is compiled each occurrence of a symbolic constant is replaced by its actual value.
- A symbolic constant is defined at the beginning of a program using the **#define** feature.
- The # define feature is called a preprocessor directive, more about the C preprocessor in a later article. A symbolic constant definition never ends with a semi colon as it is not a C statement rather it is a directive, for example:
- ```
#define PI 3.1415 //PI is the constant that will represent value 3.1415
#define True 1
#define name "Alice"
```

Prog. in C - Dr Chandran Saravanan, NITDGP

## Program Structure

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

```
#include <stdio.h>
int main() {
    /* my first program in C */
    printf("Hello, World! \n");
    return 0;
}
```

Prog. in C - Dr Chandran Saravanan, NITDGP

## extern static register auto

- **extern**  
extends the visibility of the variables
- **static**  
preserves value after scope of the variable
- **register**  
faster than other variable
- **auto**  
default storage class

Prog. in C - Dr. Chandran Saravanan, NITDGP

## Static Example

```
#include<stdio.h>
int fun(){
    static int count;
    count++;
    return count;
}
int main(){
    printf("%d ", fun());
    printf("%d ", fun());
    return 0;
}
```

Prog. in C - Dr. Saravanan Chandran, NITDGP

## Backslash codes

- **\b**      backspace
- **\f**      formfeed
- **\n**      newline
- **\r**      carriagereturn
- **\t**      horizontal tab
- **\v**      vertical tab
- **\?**      question mark

Prog. in C - Dr. Chandran Saravanan, NITDGP

## Arithmetic operators

- +**      addition
- subtraction
- \***      multiplication
- /**      division
- %**      remainder
- ++**    increment by 1
- decrement by 1

Prog. in C - Dr. Chandran Saravanan, NITDGP



## Relational operators

- > greater than
- < lesser than
- >= greater than or equal to
- <= lesser than or equal to
- == equal to
- != not equal to

Prog. in C - Dr Chandran Saravanan, NITDGP

## Logical operators

- && and
- || or
- ! Not

Truth Table

| A | B | A && B | A    B | !A |
|---|---|--------|--------|----|
| 0 | 0 | 0      | 0      | 1  |
| 0 | 1 | 0      | 1      | 1  |
| 1 | 1 | 1      | 1      | 0  |
| 1 | 0 | 0      | 1      | 0  |

Prog. in C - Dr Chandran Saravanan, NITDGP

## Bitwise operators

- & and
- | or
- ^ Exclusive OR
- ~ one's complement
- >> shift right
- << shift left

Prog. in C - Dr Chandran Saravanan, NITDGP

## ?: operator

- Exp 1 ? exp2 : exp3;
- if exp1 is true exp2 is evaluated
- otherwise exp3 is evaluated
- the final result is stored in exp1

Example

```
x=10;
y=x>9?100:200;
```

Prog. in C - Dr Chandran Saravanan, NITDGP

## & and \* operator

- Pointer operators
- access memory address and contents
- & address of
- \* content of

Example

```
int a, b, *c;
b=10;
c=&b;
a=*c;
```

Prog. in C - Dr Chandran Saravanan, NITDGP

## sizeof operator

- returns length in bytes

```
printf ("%d",sizeof(int));
```

```
float=b;
printf ("%d",sizeof(b));
```

Prog. in C - Dr Chandran Saravanan, NITDGP

## Operators 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 |

Prog. in C - Dr Chandran Saravanan, NITDGP

## Enumerations

- custom data type used for storing constant integer values and referring to them by names
- Defining enumerations

```
enum fruit {grape, cherry, lemon, kiwi};
```

grape, cherry, lemon, and kiwi,  
whose values are, by default, 0, 1, 2, and 3

Prog. in C - Dr Chandran Saravanan, NITDGP

```
enum more_fruit {banana = -17, apple, blueberry,
mango};
```

banana to be -17,  
and the remaining values are incremented by 1  
apple is -16, blueberry is -15, and mango is -14.

```
enum yet_more_fruit {kumquat, raspberry, peach,
plum = peach + 2};
```

kumquat is 0, raspberry is 1, peach is 2, and plum is 4.

Prog. in C - Dr Chandran Saravanan, NITDGP

## Declaring Enumerations

```
enum fruit {banana, apple, blueberry, mango};
enum fruit my_fruit;
```

```
enum fruit {banana, apple, blueberry, mango}
my_fruit;
```

```
enum fruit {banana, apple, blueberry, mango};
banana = 15; /* You can't do this! */
```

Prog. in C - Dr Chandran Saravanan, NITDGP

## Data Input and Output

- An input can be given in the form of a file or from the command line.
- Output means to display some data on screen, printer, or in any file.
- C programming treats all the devices as files.
- stdin
- stdout
- stderr

Prog. in C - Dr Chandran Saravanan, NITDGP

## getchar() and putchar()

- The int getchar(void) function reads the next available character from the screen and returns it as an integer.
- This function reads only single character at a time.
- The int putchar(int c) function puts the passed character on the screen and returns the same character.
- This function puts only single character at a time.

Prog. in C - Dr Chandran Saravanan, NITDGP

```
#include <stdio.h>
int main() {
    int c;
    printf( "Enter a value :");
    c = getchar();
    printf( "\nYou entered: ");
    putchar( c);
    return 0;}

```

Prog. in C - Dr Chandran Saravanan, NITDGP

## gets() and puts()

- The char \*gets(char \*s) function reads a line from stdin into the buffer pointed to by s until either a terminating newline or EOF (End of File).
- The int puts(const char \*s) function writes the string 's' and 'a' trailing newline to stdout.

```
#include <stdio.h>
int main() {
    char str[100];
    printf( "Enter a value :");
    gets( str );
    printf( "\nYou entered: ");
    puts( str ); return 0;}

```

Prog. in C - Dr Chandran Saravanan, NITDGP

## scanf() and printf()

- The int scanf(const char \*format, ...) function reads the input from the standard input stream stdin and scans that input according to the format provided.
- The int printf(const char \*format, ...) function writes the output to the standard output stream stdout and produces the output according to the format provided.
- The format can be a simple constant string, but you can specify %s, %d, %c, %f, etc., to print or read strings, integer, character or float respectively. There are many other formatting options available which can be used based on requirements.

Prog. in C - Dr Chandran Saravanan, NITDGP

```
#include <stdio.h>
int main()
{
    char str[100];
    int i;
    printf( "Enter a value :");
    scanf("%s %d", str, &i);
    printf( "\nYou entered: %s %d ", str, i);
    return 0;
}

```

Prog. in C - Dr Chandran Saravanan, NITDGP

## Selection statements

```
if (expression) statement1;
else statement2;
```

```
switch (expression){
    case constant1:
        statements;
        break;
    default statement;
}
```

Prog. In C - Dr Chandran Saravanan, NITDGP

```
if (expression) statement;
or
if (expression) { Block of statements; }
or
if (expression) { Block of statements; }
else { Block of statements; }
or
if (expression) { Block of statements; }
else if (expression) { Block of statements; }
else { Block of statements; }
```

Prog. In C - Dr Chandran Saravanan, NITDGP

```
switch( expression )
{
case constant-expression1: statements1;
[case constant-expression2: statements2;]
[case constant-expression3: statements3;]
[default : statements4;]
}
```

Prog. In C - Dr Chandran Saravanan, NITDGP

## return, break, exit, goto statement

- return from a function
- break a case in the switch or termination of a loop
- exit out of a program
- continue takes to the next iteration of the loop
- goto a particular statement using label

Prog. In C - Dr Chandran Saravanan, NITDGP

## Iteration statements

```
for (initialization; condition; increment)
    statement;

while (condition) statement;

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

Prog. in C - Dr Chandran Saravanan, NITDGP

## Function

- input
- process
- output

```
return-type function_name (parameter-list)
{
    statements ;
    return value;
}
```

Prog. in C - Dr Chandran Saravanan, NITDGP

## Example

```
int add_values (int x, int y)
{
    return x + y;
}

int add_values (x, y)
int x, int y;
{
    return x + y;
}
```

Prog. in C - Dr Chandran Saravanan, NITDGP

## Calling Functions

```
function-name (parameters);

add_values (5,4);

int a=3,b=6;
add_values (a, b);

printf("%d", add_values (5,4));
```

Prog. in C - Dr Chandran Saravanan, NITDGP

## Function Parameters

a literal value, or a value stored in variable, an address in memory, or a more complex expression built by combining these

```
int foo (int a) { a = 2 * a; return a; }
x = foo (x);
```

```
void foo (int *x) { *x = *x + 42; }
int a = 15; foo (&a);
```

Prog. in C - Dr Chandran Saravanan, NITDGP

## The main Function

- Every program requires at least one function, called 'main'. This is where the program begins executing.
- You do not need to write a declaration or prototype for main, but you do need to define it.
- The return type for main is always int. You do not have to specify the return type for main, but you can.
- However, you *cannot* specify that it has a return type other than int.

Prog. in C - Dr Chandran Saravanan, NITDGP

- the return value from main indicates the program's *exit status*.
- A value of zero or EXIT\_SUCCESS indicates success and EXIT\_FAILURE indicates an error.
- Reaching the } at the end of main without a return, or executing a return statement with no value (that is, return;) are both equivalent.

```
int main (void)
{
    puts ("Hi there!");
    return 0;
}
```

Prog. in C - Dr Chandran Saravanan, NITDGP

## Example

```
int main (int argc, char *argv[])
{
    int counter;
    for (counter = 0; counter < argc; counter++)
        printf ("%s\n", argv[counter]);
    return 0;
}
```

Prog. in C - Dr Chandran Saravanan, NITDGP

## Recursive Functions

- a function that calls itself

```
int factorial (int x)
{
    if (x < 1) return 1;
    else return (x * factorial (x - 1));
}
```

Prog. in C - Dr Chandran Saravanan, NITDGP

## Infinitely Recursive

```
int watermelon (int x)
{
    return (watermelon (x));
}
```

Prog. in C - Dr Chandran Saravanan, NITDGP

## Static Functions

- callable only within the source file where it is defined

```
static int foo (int x)
{
    return x + 42;
}
```

This is useful for building a reusable library of functions and need to include some subroutines that should not be callable by the end user.

Prog. in C - Dr Chandran Saravanan, NITDGP

## Nested Functions

- can define functions within other functions

```
int factorial (int x)
{
    int factorial_helper (int a, int b)
    {
        if (a < 1) return b;
        else
            return factorial_helper ((a - 1), (a * b));
    }
    return factorial_helper (x, 1);
}
```

Prog. in C - Dr Chandran Saravanan, NITDGP



## Unions

- custom data type used for storing several variables in the same memory space
- access any of those variables at any time
- read from one of them at a time

Prog. in C - Dr Chandran Saravanan, NITDGP

## Defining Unions

- define a union using the union keyword followed by the declarations of the union's members, enclosed in braces
- declare each member of a union as declaring a variable

```
union numbers
{
    int i;
    float f;
};
```

Prog. in C - Dr Chandran Saravanan, NITDGP

## Declaring Union Variables at Definition

```
union numbers
{
    int i;
    float f;
} first_number, second_number;
```

Prog. in C - Dr Chandran Saravanan, NITDGP

## Declaring Union Variables After Definition

```
union numbers
{
    int i;
    float f;
};
union numbers first_number, second_number;
```

Prog. in C - Dr Chandran Saravanan, NITDGP

## Initializing Union Members

```
union numbers
{
    int    i;
    float  f;
};
union numbers first_number = { 5 };
union numbers first_number = { f: 3.14159 };
union numbers first_number = { .f = 3.14159 };
```

Prog. in C - Dr Chandran Saravanan, NITDGP

## Accessing Union Members

```
union numbers
{
    int    i;
    float  f;
};
union numbers first_number;
first_number.i = 5;
first_number.f = 3.9;
```

Prog. in C - Dr Chandran Saravanan, NITDGP

## Union Practical Applications

- Online Payment
  - Credit Card
  - Debit Card
  - Net Banking
  - Wallet
- Dress Size
  - XXL
  - XL
  - L
  - M
  - S

Prog. in C - Dr Chandran Saravanan, NITDGP

## Structures

A programmer-defined data type made up of variables of other data types

### Defining Structures

define a structure using the struct keyword followed by the declarations of the structure's members, enclosed in braces.

```
struct point
{
    int x, y;
};
```

Prog. in C - Dr Chandran Saravanan, NITDGP

## Declaring Structure Variables

```
struct point
{
    int x, y;
} first_point, second_point;
```

- Declaring Structure Variables after Definition

```
struct point
{
    int x, y;
};
struct point first_point, second_point;
```

Prog. in C - Dr. Chandran Saravanan, NITDGP

## Initializing Structure Members

```
struct point
{
    int x, y;
};
struct point first_point = { 5, 10 };
```

```
struct point first_point = { y = 10, x = 5 };
```

```
struct point first_point = { y: 10, x: 5 };
```

```
struct point
{
    int x, y;
} first_point = { 5, 10 };
```

Prog. in C - Dr. Chandran Saravanan, NITDGP

```
struct pointy
{
    int x, y;
    char *p;
};
struct pointy first_pointy = { 5 };
Here, x is initialized with 5, y is initialized with 0,
struct point
{
    int x, y;
};
struct rectangle
{
    struct point top_left, bottom_right;
};
struct rectangle my_rectangle = { {0, 5}, {10, 0} };
```

Prog. in C - Dr. Chandran Saravanan, NITDGP

## Accessing Structure Members

```
first_point.x = 0;
```

```
first_point.y = 5;
```

```
my_rectangle.top_left.x = 0;
```

```
my_rectangle.top_left.y = 5;
```

```
my_rectangle.bottom_right.x = 10;
```

```
my_rectangle.bottom_right.y = 0;
```

Prog. in C - Dr. Chandran Saravanan, NITDGP

## Unions

Common storage space for all members

Occupies lower memory space

Can access only one member of union at a time

## Structures

Individual storage space for each members

Occupies higher memory space

Can access all members of structure at a time

Prog. in C - Dr Chandran Saravanan, NITDGP

## Arrays

- Can store one or more elements consecutively in memory
- Array elements are indexed beginning at position zero, not one
- The number of elements in an array must be positive
- An array is declared by specifying the data type, name, and number of elements  

```
int my_array[10];
```

Prog. in C - Dr Chandran Saravanan, NITDGP

## Zero-length arrays

- The number of elements can be zero
- Zero-length arrays are used for a variable-length object

```
struct line
{
    int length;
    char contents[0];
};
```

Prog. in C - Dr Chandran Saravanan, NITDGP

## Array size can be variables

```
void getstudname(int n)
{
    int i; char studname[n];
    for(i=0;i<n;i++) studname[i]=getchar();
}
void main(void)
{
    getstudname(10);
}
```

Prog. in C - Dr Chandran Saravanan, NITDGP

## Initializing Arrays

- Array elements can be initialized while declaring it by listing the initializing values, separated by commas, in a set of braces.

```
void main(void)
{
    int i, my_array[5] = { 0, 1, 2, 3, 4 };
    for(i=0;i<5;i++)
        printf("%d\n",my_array[i]);
}
```

Prog. in C - Dr. Chandran Saravanan, NITDGP

- Don't have to explicitly initialize all of the array elements.
- For example, this code initializes the first three elements as specified, and then initializes the last two elements to a default value of zero:

```
void main(void)
{
    int i, my_array[5] = { 0, 1, 2 };
    for(i=0;i<5;i++)
        printf("%d\n",my_array[i]);
}
```

Prog. in C - Dr. Chandran Saravanan, NITDGP

- Initialize array elements by specifying array indices and optionally the assignment operator, before the value.

```
int my_array[5] = { [2] 5, [4] 9 };
```

- or,

```
int my_array[5] = { [2] = 5, [4] = 9 };
```

- Both of those examples are equivalent to:

```
int my_array[5] = { 0, 0, 5, 0, 9 };
```

Prog. in C - Dr. Chandran Saravanan, NITDGP

- Can initialize a range of elements to the same value, by specifying the first and last indices, in the form *[first] ... [last]* .

```
int my_array[10] = { [0 ... 3] = 1, [4 ... 8] = 2, 3 };
```

- That initializes elements 0 through 3 to 1, elements 4 through 8 to 2, and element 9 to 3. (also could explicitly write `[9] = 3`.)
- Notice that *must* have spaces on both sides of the `'...'`

Prog. in C - Dr. Chandran Saravanan, NITDGP

- If every element of an array is initialized, then its size is determined by the number of elements initialized.

```
int my_array[] = { 0, 1, 2, 3, 4 };
```

- If specified which elements to be initialized, then the size of the array is equal to the highest element number initialized, plus one.

```
int my_array[] = { 0, 1, 2, [9] = 9 };
```

- In that example, only four elements are initialized, but the last one initialized is element number 9, so there are 10 elements.

Prog. in C - Dr Chandran Saravanan, NITDGP

## Accessing Array Elements

- Specifying the array name, followed by the element index, enclosed in brackets.

```
my_array[0] = 5;
```

- That assigns the value 5 to the first element in the array, at position zero.

Prog. in C - Dr Chandran Saravanan, NITDGP

## Multidimensional Arrays

- Arrays of Arrays - A two-dimensional array that holds five elements in each dimension

```
int two_dimensions[2][5] = { { 1, 2, 3, 4, 5 }, { 6, 7, 8, 9, 10 } };
```

- `two_dimensions[1][3] = 12;`

Prog. in C - Dr Chandran Saravanan, NITDGP

```
# include <stdio.h>
```

```
void main(void)
```

```
{
```

```
int i,j, two_dimensions[2][5] = { { 1, 2, 3, 4, 5 }, { 6, 7, 8, 9, 10 } };
```

```
for(i=0;i<2;i++)
```

```
    for (j=0;j<5;j++)
```

```
        printf("%d\n",two_dimensions[i][j]);
```

```
}
```

Prog. in C - Dr Chandran Saravanan, NITDGP

## Arrays as strings

An array of characters to hold a string

```
char blue[26];
char yellow[26] = {'y', 'e', 'l', 'l', 'o', 'w', '\0'};
char orange[26] = "orange";
char gray[] = {'g', 'r', 'a', 'y', '\0'};
char salmon[] = "salmon";
```

Prog. in C - Dr Chandran Saravanan, NITDGP

The string terminator null character `\0` is included at the end of the string

```
char lemon[26] = "custard";
lemon = "steak sauce"; /* Fails! */
Can change one character at a time
char name[] = "bob";
name[0] = 'r';
```

Prog. in C - Dr Chandran Saravanan, NITDGP

- It is possible to initialize an array using a string that has more characters than the specified size.
- This is not a good thing. The larger string will *not* override the previously specified size of the array, and you will get a compile-time warning.
- Since the original array size remains, any part of the string that exceeds that original size is being written to a memory location that was not allocated for it.

```
char greet[5]="Good Morning";
```

Prog. in C - Dr Chandran Saravanan, NITDGP

## Arrays of Unions

```
union numbers
{
    int i;
    float f;
};
union numbers number_array [3];
union numbers number_array [3] = { {3}, {4}, {5} };
union numbers number_array [3];
number_array[0].i = 2;
```

Prog. in C - Dr Chandran Saravanan, NITDGP

## Arrays of Structures

- Array of a structure type is created as an array of a primitive data type.

```
struct point
{
    int x, y;
};
```

```
struct point point_array [3];
```

- Created a 3-element array of struct point variables called point\_array.

Prog. in C - Dr Chandran Saravanan, NITDGP

- Initialize the elements of a structure array
- ```
struct point point_array [3] = { {2, 3}, {4, 5}, {6, 7} };
```

- The additional braces are used for partially initializing some of the structures in the array, and fully initialize others:

```
struct point point_array [3] = { {2}, {4, 5}, {6, 7} };
```

- The value 4 is assigned to the x member of the second array element, *not* to the y member of the first element, as would be the case without the grouping braces.

Prog. in C - Dr Chandran Saravanan, NITDGP

- Access the structure members in the array using the member access operator.

```
struct point point_array [3];
```

```
point_array[0].x = 2;
```

```
point_array[0].y = 3;
```

Prog. in C - Dr Chandran Saravanan, NITDGP

## Typedef

- to give a type, a new name

```
typedef unsigned char BYTE;
```

- the identifier BYTE can be used as an abbreviation for the type unsigned char

```
BYTE b1, b2;
```

Prog. in C - Dr Chandran Saravanan, NITDGP



## Preprocessors

- The C Preprocessor is not a part of the compiler, but is a separate step in the compilation process.
- A C Preprocessor is just a text substitution tool and it instructs the compiler to do required pre-processing before the actual compilation.
- All preprocessor commands begin with a hash symbol (#).
- It must be the first nonblank character, and for readability, a preprocessor directive should begin in the first column.

Prog. in C - Dr. Chandran Saravanan, NITDGP

### Directive

### Description

<b>#define</b>	Substitutes a preprocessor macro.
<b>#include</b>	Inserts a particular header from another file.
<b>#undef</b>	Undefines a preprocessor macro.
<b>#ifdef</b>	Returns true if this macro is defined.
<b>#ifndef</b>	Returns true if this macro is not defined.
<b>#if</b>	Tests if a compile time condition is true.
<b>#else</b>	The alternative for #if.
<b>#elif</b>	#else and #if in one statement.
<b>#endif</b>	Ends preprocessor conditional.
<b>#error</b>	Prints error message on stderr.
<b>#pragma</b>	Issues special commands to the compiler, using a standardized method.

Prog. in C - Dr. Chandran Saravanan, NITDGP

## Examples

```
#define MAX_ARRAY_LENGTH 20
#include <stdio.h>
#include "myheader.h"
#undef FILE_SIZE
#define FILE_SIZE 42
#ifndef MESSAGE
#define MESSAGE "You wish!"
#endif
#ifdef DEBUG
/* Your debugging statements here */
#endif
```

Prog. in C - Dr. Chandran Saravanan, NITDGP

## Preprocessor Operators

- The Macro Continuation (\) Operator

```
#define message_for(a, b) \
printf("#a " and " #b ": We love you!\n")
```

- The Stringize (#) Operator

```
#include <stdio.h>
#define message_for(a, b) \
printf("#a " and " #b ": We love you!\n")
int main(void) {
    message_for(Carole, Debra);
    return 0;
}
Carole and Debra: We love you!
```

Prog. in C - Dr. Chandran Saravanan, NITDGP

## The Token Pasting (##) Operator

```
#include <stdio.h>

#define tokenpaster(n) printf ("token" #n " = %d",\
token##n)

int main(void) {
    int token34 = 40;
    tokenpaster(34);
    return 0;
}
token34 = 40
```

Prog. in C - Dr Chandran Saravanan, NITDGP

## The Defined() Operator

```
#include <stdio.h>
#if !defined (MESSAGE)
    #define MESSAGE "You wish!"
#endif

int main(void) {
    printf("Here is the message: %s\n", MESSAGE);
    return 0;
}
Here is the message: You wish!
```

- **Parameterized Macros**

```
#define square(x) ((x) * (x))
```

Prog. in C - Dr Chandran Saravanan, NITDGP

## Predefined Macros

Macro	Description
<code>__DATE__</code>	The current date as a character literal in "MMM DD YYYY" format.
<code>__TIME__</code>	The current time as a character literal in "HH:MM:SS" format.
<code>__FILE__</code>	This contains the current filename as a string literal.
<code>__LINE__</code>	This contains the current line number as a decimal constant.
<code>__STDC__</code>	Defined as 1 when the compiler complies with the ANSI standard.

Prog. in C - Dr Chandran Saravanan, NITDGP

```
#include <stdio.h>
main() {
    printf("File: %s\n", __FILE__);
    printf("Date: %s\n", __DATE__);
    printf("Time: %s\n", __TIME__);
    printf("Line: %d\n", __LINE__);
    printf("ANSI: %d\n", __STDC__);
}
```

- File :test.c
- Date :Jun 2 2012
- Time :03:36:24
- Line :8
- ANSI :1

Prog. in C - Dr Chandran Saravanan, NITDGP

## Pointers

- Pointers hold memory addresses of stored constants or variables.

### Declaring Pointers

*data-type* \* *name*;

- White space is not significant around the indirection operator:

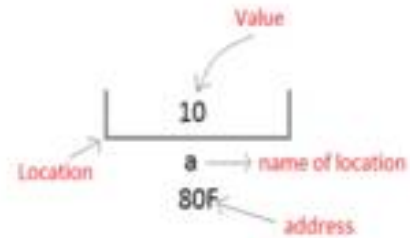
*data-type* \* *name*;

*data-type* \* *name*;

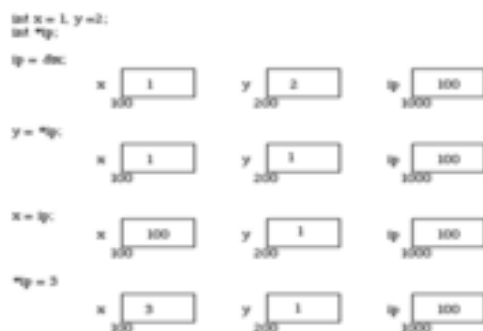
```
int *foo, *bar; /* Two pointers. */
```

```
int *baz, quux; /* A pointer and an integer variable. */
```

Prog. in C - Dr. Chandran Saravanan, NITDGP



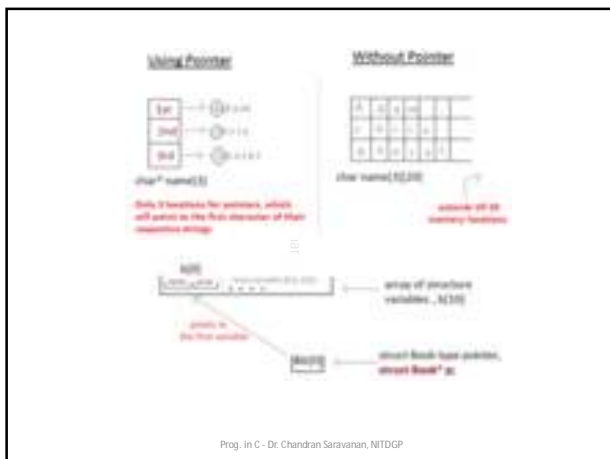
Prog. in C - Dr. Chandran Saravanan, NITDGP



Prog. in C - Dr. Chandran Saravanan, NITDGP



Prog. in C - Dr. Chandran Saravanan, NITDGP



## Initializing Pointers

```
int i;
int *ip = &i;
# include <stdio.h>
void main(void){
    int i=3, j=4, *ip, *jp;
    ip = &i; jp = &j;
    printf("%u\t%u\t%u\t%u", i, j, ip, jp);
    printf("%u\t%u\t%u\t%u", i, j, *ip, *jp);
}

int i, j; int *ip = &i; /* 'ip' now holds the address of 'i'. */
ip = &j; /* 'ip' now holds the address of 'j'. */
*ip = &i; /* 'j' now holds the address of 'i'. */
```

## Pointers to Unions

- A pointer to a union type is created as a pointer to a primitive data type.

```
union numbers
{
    int i;
    float f;
};
union numbers foo = {4};
union numbers *number_ptr = &foo;
```

- Access the members of a union variable through a pointer

```
number_ptr -> i = 450;
```

## Pointers to Structures

- A pointer to a structure type is created as a pointer to a primitive data type.

```
struct fish
{
    float length, weight;
};
struct fish salmon = {4.3, 5.8};
struct fish *fish_ptr = &salmon;
```

- Access the members of a structure variable through a pointer

```
fish_ptr -> length = 5.1;
fish_ptr -> weight = 6.2;
```

## NULL Pointer

- Good practice to assign a NULL value to a pointer variable in case you do not have an exact address to be assigned.
- This is done at the time of variable declaration.
- A pointer that is assigned NULL is called a **null** pointer.

• `int *ptr = NULL;`

- To check for a null pointer, use an 'if' statement as follows

• `if(ptr) /* succeeds if p is not null */`  
 • `if(!ptr) /* succeeds if p is null */`

Prog. in C - Dr Chandran Saravanan, NITDGP

## Pointer arithmetic operations

- Increment `++`
- Decrement `--`
- Addition `+`
- Subtraction `-`
- `ptr++;`
- `ptr--;`
- `ptr=ptr+1;`
- `ptr=ptr-1;`

Prog. in C - Dr Chandran Saravanan, NITDGP

## Memory Allocation

- `malloc()` allocates requested size of bytes and returns a pointer first byte of allocated space

`ptr = (cast-type*) malloc(byte-size)`

`ptr = (int*) malloc(100 * sizeof(int));`

- `calloc()` allocates space for an array elements, initializes to zero and then returns a pointer

`ptr = (cast-type*) calloc(n, element-size);`

`ptr = (float*) calloc(25, sizeof(float));`

Prog. in C - Dr Chandran Saravanan, NITDGP

- `free()` Deallocate the previously allocated space

`free(ptr);`

- `realloc()` Change the size of previously allocated space

`ptr = realloc(ptr, newsize);`

`ptr = realloc(ptr, 40);`

Prog. in C - Dr Chandran Saravanan, NITDGP

## Swap two numbers using pointers

```
#include <stdio.h>
void swap(int *n1, int *n2);
int main() {
    int num1 = 5, num2 = 10;
    swap(&num1, &num2); // call by reference
    printf("Number1 = %d\n", num1);
    printf("Number2 = %d", num2);
    return 0;
}
void swap(int *n1, int *n2)
{
    int temp;
    temp = *n1;
    *n1 = *n2;
    *n2 = temp;
}
```

Prog. in C - Dr. Chandran Saravanan, NITDGP

## Return value as pointer

```
#include <stdio.h>
#include <conio.h>
int* larger(int*, int*);
void main()
{
    int a=15;
    int b=92;
    int *p;
    p=larger(&a, &b);
    printf("%d is larger", *p);
}
int* larger(int *x, int *y)
{
    if(*x > *y) return x;
    else return y;
}
```

Prog. in C - Dr. Chandran Saravanan, NITDGP

## Multiple indirection

```
int a = 3;
int *b = &a;
int **c = &b;
int ***d = &c;

***d == **c == *b == a == 3;
```

Prog. in C - Dr. Chandran Saravanan, NITDGP

## Constant Pointers

- These two declarations are equivalent

```
const int *ptr_a;    int const *ptr_b;
```

- These two declarations are not equivalent

```
int const *ptr_c;
```

- you cannot change \*ptr\_c = 42; you can \*ptr\_c++;

```
int *const ptr_d;
```

- you can change \*ptr\_d = 42; you cannot ptr\_d++;

Prog. in C - Dr. Chandran Saravanan, NITDGP

## Data Files

- File is a place on your Disk (secondary memory-permanent) where information is stored.
- Storing in a file will preserve your data even if the program terminates.
- One time entry of large number of data
- Moving data from one computer to another

Prog. in C - Dr Chandran Saravanan, NITDGP

## Types of Files

- There are two types of files.
- **1. Text files**
  - Text files are the normal .txt files that can be easily created using Notepad or any text editors. The contents of the file is plain text and visible. Edit or delete the contents easily. They take minimum effort to maintain, are easily readable, and provide least security and takes bigger storage space.
- **2. Binary files**
  - Binary files are mostly the .bin files in your computer. Instead of storing data in plain text, they store it in the binary form (0's and 1's). They can hold higher amount of data, are not readable easily and provides a better security than text files.

Prog. in C - Dr Chandran Saravanan, NITDGP

## File Operations

- There are six major operations on the text or binary file
1. Creating a new file
  2. Opening an existing file
  3. Closing a file
  4. Reading information from a file
  5. Writing information to a file
  6. Seeking information in a file

Prog. in C - Dr Chandran Saravanan, NITDGP

## Working with files

- Declare a pointer of type file; this declaration is needed for communication between the file and program.

```
FILE *fptr;
```

Prog. in C - Dr Chandran Saravanan, NITDGP

## Opening a file - for creation and edit

- The syntax for opening a file in standard I/O is:

- `ptr = fopen("filename", "mode")`

- For Example:

- `fopen("E:\\cprogram\\newprogram.txt", "w");`
- `fopen("E:\\cprogram\\oldprogram.bin", "rb");`

Prog. in C - Dr Chandran Saravanan, NITDGP

- Let's suppose the file `newprogram.txt` doesn't exist in the location `E:\\cprogram`. The first function creates a new file named `newprogram.txt` and opens it for writing as per the mode `'w'`. The writing mode allows you to create and edit (overwrite) the contents of the file.

- Now let's suppose the second binary file `oldprogram.bin` exists in the location `E:\\cprogram`. The second function opens the existing file for reading in binary mode `'rb'`. The reading mode only allows you to read the file, you cannot write into the file.

Prog. in C - Dr Chandran Saravanan, NITDGP

File Mode	Meaning of Mode	During Inexistence of file
<b>r</b>	Open for reading.	If the file does not exist, <code>fopen()</code> returns <code>NULL</code> .
<b>rb</b>	Open for reading in binary mode.	If the file does not exist, <code>fopen()</code> returns <code>NULL</code> .
<b>w</b>	Open for writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
<b>wb</b>	Open for writing in binary mode.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
<b>a</b>	Open for append.	If the file does not exist, it will be created.
<b>ab</b>	Open for append in binary mode.	If the file does not exist, it will be created.

Prog. in C - Dr Chandran Saravanan, NITDGP

<b>r+</b>	Open for both reading and writing.	If the file does not exist, <code>fopen()</code> returns <code>NULL</code> .
<b>rb+</b>	Open for both reading and writing in binary mode.	If the file does not exist, <code>fopen()</code> returns <code>NULL</code> .
<b>w+</b>	Open for both reading and writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
<b>wb+</b>	Open for both reading and writing in binary mode.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
<b>a+</b>	Open for both reading and appending.	If the file does not exist, it will be created.
<b>ab+</b>	Open for both reading and appending in binary mode.	If the file does not exist, it will be created.

Prog. in C - Dr Chandran Saravanan, NITDGP



## Closing a File

- The file (both text and binary) should be closed after reading/writing. Closing a file is performed using library function `fclose()`.
- `fclose(fptr);`

Prog. in C - Dr Chandran Saravanan, NITDGP

## Write to a text file using `fprintf()`

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int num;
    FILE *fptr;
    fptr = fopen("C:\\program.txt", "w");
    if (fptr == NULL)
    {
        printf("Error!");
        exit(1);
    }
    printf("Enter num: ");
    scanf("%d", &num);
    fprintf(fptr, "%d", num);
    fclose(fptr);
    return 0;
}
```

Prog. in C - Dr Chandran Saravanan, NITDGP

## Read from a text file using `fscanf()`

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int num;
    FILE *fptr;
    if ((fptr = fopen("C:\\program.txt", "r")) == NULL) {
        printf("Error! opening file");
        // Program exits if the file pointer returns NULL.
        exit(1);
    }
    fscanf(fptr, "%d", &num);
    printf("Value of n=%d", num);
    fclose(fptr);
    return 0;
}
```

Prog. in C - Dr Chandran Saravanan, NITDGP

## Writing to a binary file

```
#include <stdio.h>
#include <stdlib.h>
struct threeNum {
    int n1, n2, n3;
};
int main() {
    int n;
    struct threeNum num;
    FILE *fptr;
    if ((fptr = fopen("C:\\program.bin", "wb")) == NULL) {
        printf("Error! opening file");
        exit(1); // Program exits if the file pointer returns NULL.
    }
    for (n = 1; n < 5; ++n) {
        num.n1 = n;
        num.n2 = 5*n;
        num.n3 = 5*n + 1;
        fwrite(&num, sizeof(struct threeNum), 1, fptr);
    }
    fclose(fptr);
    return 0;
}
```

Prog. in C - Dr Chandran Saravanan, NITDGP

## Reading from a binary file

```
#include <stdio.h>
#include <stdlib.h>
struct threeNum{
    int n1, n2, n3;
};
int main(){
    int n;
    struct threeNum num;
    FILE *fptr;
    if ((fptr = fopen("C:\\program.bin", "rb")) == NULL){
        printf("Error! opening file");
        exit(1); // Program exits if the file pointer returns NULL.
    }
    for(n = 1; n < 5; ++n) {
        fread(&num, sizeof(struct threeNum), 1, fptr);
        printf("n1: %d\t n2: %d\t n3: %d", num.n1, num.n2, num.n3);
    }
    fclose(fptr);
    return 0;
}
```

Prog. in C - Dr Chandran Saravanan, NITDGP

## Seeking the cursor in the file

fseek(FILE \* stream, long int offset, int whence)

### Whence Meaning

SEKK\_SET offset starts from beginning of the file  
 SEKK\_END offset starts from end of the file  
 SEKK\_CUR offset starts from current location of the file.

- ftell() - It tells the byte location of current position of cursor in file pointer.
- rewind() - It moves the control to beginning of the file.

```
printf("n1: %d\t n2: %d\t n3: %d\n", num.n1, num.n2, num.n3, ftell(fptr));
rewind(fptr);
```

- EOF - end of file

if (feof(fp)) printf("\n End of file reached.");

Prog. in C - Dr Chandran Saravanan, NITDGP

```
#include <stdio.h>
#include <stdlib.h>
struct threeNum{
    int n1, n2, n3;
};
int main(){
    int n;
    struct threeNum num;
    FILE *fptr;
    if ((fptr = fopen("C:\\program.bin", "rb")) == NULL){
        printf("Error! opening file"); // Program exits if the file pointer returns NULL.
        exit(1);
    }
    fseek(fptr, sizeof(struct threeNum), SEEK_END); // Moves the cursor to the end of the file
    for(n = 1; n < 5; ++n) {
        fread(&num, sizeof(struct threeNum), 1, fptr);
        printf("n1: %d\t n2: %d\t n3: %d", num.n1, num.n2, num.n3);
    }
    fclose(fptr);
    return 0;
}
```

Prog. in C - Dr Chandran Saravanan, NITDGP

## Implementation of file copy

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
void main(void)
{
    FILE *fp1, *fp2;
    char ch;
    clrscr();
    fp1 = fopen("Sample.txt", "r");
    fp2 = fopen("Output.txt", "w");
    while (1) {
        ch = fgetc(fp1);
        if (ch == EOF)
            break;
        else
            putc(ch, fp2);
    }
    printf("File copied Successfully!");
    fclose(fp1);
    fclose(fp2);
}
```

Prog. in C - Dr Chandran Saravanan, NITDGP

## Low-Level Programming

- Object code is small and efficient.
- Optimize the use of three resources:
- Execution time,
- Memory,
- Development/maintenance time.

Prog. in C - Dr Chandran Saravanan, NITDGP

## Bitwise operators

- &            and
- |            or
- ^ Exclusive OR
- ~ one's complement
- >> shift right
- << shift left

Prog. in C - Dr Chandran Saravanan, NITDGP

```
#include <stdio.h>
main() {
    unsigned int a = 60; /* 60 = 0011 1100 */
    unsigned int b = 13; /* 13 = 0000 1101 */
    int c = 0;
    c = a & b; /* 12 = 0000 1100 */
    printf("Line 1 - Value of c is %d\n", c);
    c = a | b; /* 61 = 0011 1101 */
    printf("Line 2 - Value of c is %d\n", c);
    c = a ^ b; /* 49 = 0011 0001 */
    printf("Line 3 - Value of c is %d\n", c);
    c = ~a; /* 61 = 1100 0011 */
    printf("Line 4 - Value of c is %d\n", c);
    c = a << 2; /* 240 = 1111 0000 */
    printf("Line 5 - Value of c is %d\n", c);
    c = a >> 2; /* 15 = 0000 1111 */
    printf("Line 6 - Value of c is %d\n", c);
}
```

Prog. in C - Dr Chandran Saravanan, NITDGP

## 'C' Function Exercises

1. A function called `abs_val` that returns `int` and takes an `int` argument. It returns the absolute value of its argument, by negating it if it is negative.
2. Write a C program to find cube of any number using function.
3. Write a C program to find diameter, circumference and area of circle using functions. (input is radius)
4. Write a C program to find maximum and minimum between two numbers using functions.
5. Write a C program to check whether a number is even or odd using functions.

Prog. in C - Dr Chandran Saravanan, NITDGP

6. Write a C program to find all prime numbers between given interval using functions.
7. Write a C program to print all strong numbers between given interval using functions. (Strong numbers are the numbers whose sum of factorial of digits is equal to the original number. Example: 145 is a strong number)
8. Write a C program to print all armstrong numbers between given interval using functions. (sum of the cubes of its digits is equal to the number itself)
9. Write a C program to print all perfect numbers between given interval using functions. (a perfect number is a positive integer equal to the sum of its proper positive divisors, that is, the sum of its positive divisors excluding the number itself)
10. Write a C program to find power of any number using recursion

Prog. in C - Dr Chandran Saravanan, NITDGP

11. Write a C program to print all natural numbers between 1 to n using recursion.
12. Write a C program to find sum of all natural numbers between 1 to n using recursion.
13. Write a C program to find reverse of any number using recursion.
14. Write a C program to check whether a number is palindrome or not using recursion.
15. Write a C program to find sum of digits of a given number using recursion.
16. Write a C program to find factorial of any number using recursion.

Prog. in C - Dr Chandran Saravanan, NITDGP

17. Write a C program to generate  $n^{\text{th}}$  Fibonacci term using recursion.
18. Write a C program to find Greatest Common Divisor (GCD) / Highest Common Factor (HCF) of two numbers using recursion.
19. Write a C program to find Least Common Multiple (LCM) of two numbers using recursion.
20. Write a C program to display all array elements using recursion.
21. Write a C program to find sum of elements of array using recursion.
22. Write a C program to find maximum and minimum elements in array using recursion.

Prog. in C - Dr Chandran Saravanan, NITDGP

## 'C' Arrays Exercises

1. Write a program in C to read n number of values in an array and display it in reverse order.
2. Write a program in C to count a total number of duplicate / unique elements in an array.
3. Write a program in C to print all unique elements in an array.
4. Write a program in C to merge two arrays of same size sorted in ascending order.
5. Write a program in C to count the frequency of each element of an array.

Prog. in C - Dr Chandran Saravanan, NITDGP

6. Write a program in C to find the maximum and minimum element in an array.
7. Write a program in C to separate odd and even integers in separate arrays.
8. Write a program in C to sort elements of array in ascending / descending order.
9. Write a program in C to insert New value in the array (sorted list ).
10. Write a program in C to delete an element at desired position from an array.

Prog. in C - Dr Chandran Saravanan, NITDGP

11. Write a program in C to find the second largest / smallest element in an array.
12. Write a program in C for addition / subtraction / multiplication / transpose of two Matrices of same size.
13. Write a program in C to calculate determinant of a 3 x 3 matrix.
14. Write a program in C to accept two matrices and check whether they are equal.
15. Write a program in C to check whether a given matrix is an identity matrix.

Prog. in C - Dr Chandran Saravanan, NITDGP

## 'C' File Exercises

1. Write a program in C to read the file and store the lines into an array.
2. Write a program in C to Find the Number of Lines in a Text File.
3. Write a program in C to count a number of words and characters in a file.
4. Write a program in C to delete a specific line from a file.
5. Write a program in C to replace a specific word with another word in a file.
6. Write a program in C to merge two files and write it in a new file.
7. Write a program in C to encrypt and decrypt a text file.

Prog. in C - Dr Chandran Saravanan, NITDGP

## 'C' Pointer Exercises

1. Write a program in C to add two numbers using pointers.
2. Write a program in C to add numbers using call by reference.
3. Write a program in C to store n elements in an array and print the elements using pointer.
4. Write a program in C to print all permutations of a given string using pointers.
5. Write a program in C to calculate the length of the string using a pointer.

Prog. in C - Dr Chandran Saravanan, NITDGP

6. Write a program in C to swap two elements using call by reference.
7. Write a program in C to find the factorial of a given number using pointers.
8. Write a program in C to count the number of vowels and consonants in a string using a pointer.
9. Write a program in C to sort an array of numbers using Pointer.
10. Write a program in C to print a string in reverse using a pointer.

Prog. In C - Dr Chandran Saravanan, NITDGP

## PDF source of this presentation

- <http://www.slideshare.net/cs1973/prog-langc>

Saravanan Chandran

Prog. In C - Dr Chandran Saravanan, NITDGP