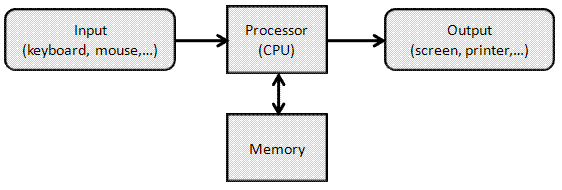
1. INTRODUCTION TO COMPUTER CONCEPTS AND REASONING, ALGORITHMS AND FLOWCHARTS

**1.1 Introduction to Computer Hardware, Software and its Types**

A computer is a machine that manipulates data according to a list of instructions called a program.



By Paul Mullins

A computer can also be defined as an electronic machine that accepts input (data), processes it and gives out results (information). A program or application program in memory manages the processing. It can make use of data or information stored in memory and remember the results of the processing for later use, again in memory.

A computer is an electronic device, operating under the control of instructions stored in its own memory that can accept data (input), process the data according to specified rules, produce information (output), and store the information for future use.

Central Processing Unit (CPU) :A CPU is brain of a computer. It is responsible for all functions and processes. Regarding computing power, the CPU is the most important element of a computer system.

The CPU is comprised of three main parts :

**Arithmetic Logic Unit (ALU):**Executes all arithmetic and logical operations.Arithmetic calculations like as addition, subtraction, multiplication and division. Logical operation like compare numbers, letters, or special characters

**Control Unit (CU):** controls and co-ordinates computer components.

1.Read the code for the next instruction to be executed.

2.Increment the program counter so it points to the next instruction.

3.Read whatever data the instruction requires from cells in memory.

4.Provide the necessary data to an ALU or register.

5.If the instruction requires an ALU or specialized hardware to complete, instruct the hardware

to perform the requested operation.

**Registers :**Stores the data that is to be executed next, "very fast storage area".

**Computer Components:**

A computer system consists of both **hardware** and information stored on hardware. Information stored on computer hardware is often called ***software***.

The **hardware** components of a computer system are the electronic and mechanical parts.

Computer hardware is the collection of physical elements that constitutes a computer system. Computer hardware refers to the physical parts or components of a computer such as the monitor, mouse, keyboard, computer data storage, hard drive disk (HDD), system unit (graphic cards, sound cards, memory, motherboard and chips), etc. all of which are physical objects that can be touched.

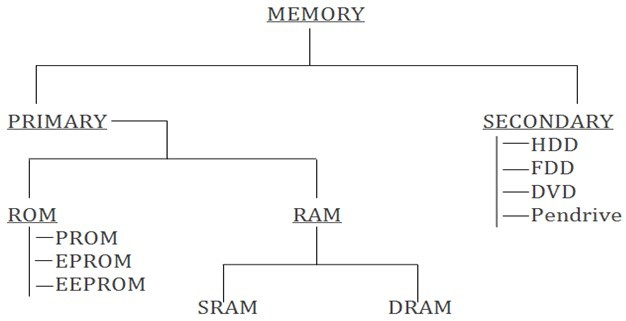
**Hardware:**

The major hardware components of a computer system are:

* Processor
* Main memory
* Secondary memory
* Input devices
* Output devices

**Memory:**

Memory is storage part in computer. It is store the data, information, programs during processing in computer. It stores data either temporarily or permanent basis. Memory used to important role in saving and retrieving data.



**Types of Memory**– Mainly computer have two types memory

1. **Primary Memory / Volatile Memory.**
2. **Secondary Memory / Non Volatile Memory.**

**1.  Primary Memory / Volatile Memory–**Primary memory is internal memory of the computer.  It is also known as main memory and Temporary memory .Primary Memory holds the data and instruction on which computer is currently working. Primary Memory by is nature volatile that is when power is switched off its data will be lost.

**Types of Primary Memory–** Primary memory is generally of two types:

1. **RAM**
2. **ROM**

**1. RAM (Random Access Memory) –** It stands for Random Access Memory.RAM is known as read /writes memory. It generally refereed as main memory of the computer system. It is a temporary memory. The information stored in this memory is lost as the power supply to the computer is switched off. That’s why RAM is also called “**Volatile Memory”**

**Types of RAM–   RAM is also of two types:**

**a) Static RAM-** Static RAM also known as SRAM ,retain stored information as long as the power supply is ON. SRAM are of higher coast and consume more power .They have higher speed than Dynamic RAM

**b)Dynamic RAM**– Dynamic RAM also known as DRAM, its stored information in a very short time (a few milliseconds) even though the power supply is ON. The Dynamic RAM are cheaper and moderate speed and also they consume less power.

**2. ROM  (Read Only Memory) –**It stands for Read Only  Memory.ROM is a Permanent Type memory. Its content are not lost when power supply is switched off. Content of ROM is decided by the computer manufacturer and permanently stored at the time of manufacturing. ROM cannot be overwritten by the computer. It is also called **“Non-Volatile Memory”.**

**Type of ROM**: ROM memory is three types names are following-

1. **PROM(Programmable Read Only Memory**)-PROM chip is programmable ROM.it is PROM chips to write data once and read many.once chip has been programmed ,the recorded information cannot be changed. PROM is also nonvolatile memory.
2. **EPROM (Erasable Programmable Read Only Memory)-** EPROM chip can be programmed time and again by erasing the information stored earlier in it. Information stored in EPROM exposing the chip for some time ultraviolet light .
3. **EEPROM (Electrically Erasable Programmable Read Only Memory)**-The EEPROM is programmed and erased by special electrical waves in millisecond. A single byte of a data or the entire contents of device can be erased.

**2. Secondary Memory / Non Volatile Memory–**

Secondary Memory is external memory of the computer. It is also known as Auxiliary memory and permanent memory. It is used to store the different programs and the information permanently.

Secondary Memory is nature non volatile. It means data is stored permanently even if power is switched off.

**The secondary storage devices are:**

1. Floppy Disks
2. Magnetic (Hard) Disk
3. Magnetic Tapes
4. Pen Drive
5. Winchester Disk
6. Optical Disk(CD,DVD)

**Input Devices :**Input device is any peripheral (piece of computer hardware equipment to provide data and control signals to an information processing system such as a computer or other information appliance. Input device Translate data from form that humans understand to one that the computer can work with. Most common are keyboard and mouse.

**Output devices:**Every computer system will need to have a way of displaying information or printing information out so that it is useful to humans. There needs to be, therefore, some ‘output devices’. A very useful output device is a VDU (or Visual Display Unit). Another handy one that allows you to produce ‘hardcopy’ (i.e. a printout on paper) is a printer.

## Software:

## Software is a set of programmes/commands designed to perform a well-defined task.

software systems into two major classes:

* **System software:** Helps run the computer hardware and computer system itself. System software includes operating systems, device drivers, diagnostic tools and more. System software is almost always pre-installed on your computer.
* **Application software:** Allows users to accomplish one or more tasks. It includes word processing, web browsing and almost any other task for which you might install software. (Some application software is pre-installed on most computer systems.)

Software is generally created /written in a high-level programming language, one that is (more or less) readable by people. These high-level instructions are converted into "machine language" instructions, represented in binary code, before the hardware can "run the code".

**Utility Software:**is system software designed to help analyze, configure, optimize or maintain a computer. Utility software, along with operating system software, is a type ofsystem software used to support the computer infrastructure, distinguishing it from application software which is aimed at directly performing tasks that benefit ordinary users**. PDF Creator, File Archiving and PDF Converters are commonly used utility software.**

1.2Arithmetic Reasoning

1. If Jean’s weekly income doubled she would be making $120 a week more than Barbara. Jean’s weekly income is $80 more than half of Betty’s. Betty makes $200 a week.

How much does Barbara make?

a. $180 c. $240

b. $200 d. $360

2. A conference with 3600 participants gathers in Albany. One of every twelve people attending the conference who have ordered meals has special dietary needs. Half of those attending the conference signed up for meals. How many have special dietary needs?

a. 266 c. 150

b. 133 d. 300

3. It costs $360 for Office X’s service contract with a typewriter company to service 18 typewriters for six months. At this same rate, how much would it cost Office X to service six typewriters for three months?

a. $80 c. $90

b. $75 d. $60

4. In December, an office spent $480, or 15% of its non-personnel expenses that month, for postage. What were its total non-personnel expenses for December?

a. $3200 c. $5520

b. $552 d. $7200

5. Catherine bought an equal number of $11.00, $9.00, and $8.00 tickets for a concert. She spent $196 for all of the tickets. How many of each did she buy?

a. 6 c. 8

b. 7 d. Cannot be determined from information given

6. Agency Y employs 13,800 people. Of these, 42% are male, and 50% of the males are age 30 or younger. How many males are there in Agency Y who are older than 30?

a. 5796 c. 3471

b. 2898 d. 2910

7. A machine can collate 126 400 page books in 14 days. If it continues to collate at this same rate, how many 400 page books could it collate in 30 days?

a. 256 c. 248

b. 290 d. 270

8. A typewriter and a dictation machine cost a total of $840. If the typewriter cost $360 more than the dictation machine, how much did the dictation machine cost?

a. $480 c. $240

b. $440 d. $280

9. A cabinet maker has a round piece of wood 1/2" in diameter and 3/4 yards long. She needs half the length for the back of a chair and the remaining piece for 3/4" pegs. How many pegs will she have?

a. 18 c. 10

b. 9 d. 8/9

10. Robin can wallpaper a room in four hours, Susan can wallpaper the same room in seven hours. How long will it take them to wallpaper the room if they work together?

a. 4.5 hours c. 5.5 hours

b. 3.2 hours d. 2.5 hours

11. Mary and Alice jog 3 miles each evening. If they run at a constant rate and it takes Mary 40 minutes while Alice finishes in half an hour, how much distance does Mary have left when Alice finishes?

a. I mile c. 2/3 mile

b. 3/4 mile d. 1.33 miles

12. As a fund raiser, a community organization buys tickets to the theatre to resell 25% above cost. They buy 50 eight dollar tickets, 25 ten dollar tickets and 25 fifteen dollar tickets. If they sell all but three of the ten dollar tickets, how much money have they made?

a. $218.75 c. $248.75

b. $1273 d. $1243.75

13. If a couch cost $640 after a 20% discount, what was its original price?

a. $768 c. $800

b. $512 d. $780

14. A salesperson traveled 145 miles Monday, 72 miles Tuesday, and 98 miles Wednesday for $2300 worth of sales. If the business pays 2l¢ a mile for gas and vehicle maintenance, approximately what percent of sales for the three days went to gas and vehicle maintenance?

a. 3% c. 6%

b. 8% d. 1%

15. If one of every eight junior year students at a high school takes Latin, approximately what percent of a junior year class of 650 took Latin?

a. 6 c. 81

b. 14 d. 13

16. The proposed budget for a new social service program is $102,000. The budget states that the project has secured $14,500 worth of transportation services and $1,200 worth of office equip- ment as inkind contributions. What percent of the budget has been secured inkind?

a. 21.4 c. 20.9

b. 15.4 d. 13.1

17. The purpose of the program in Question 16 is to distribute 250,000 pounds of food to disadvan- taged persons. Approximately how many pounds of food will be distributed for each dollar budgeted for the program?

a. 3.73 c. 2.45

b. 2.15 d. 2.95

18. A community cannery charges 15¢ per quart and 25¢ for 2 pints for processing. People using the cannery purchase can jars there at 15% discount off the regular price of $4.25 per case of a dozen quart or pint jars. How much will it cost to can 76 quarts of tomatoes and 20 pints of jelly if the jars are bought at the cannery? (Jars are not sold individually).

a. $46.39 c. $32.49

b. $37.14 d. $13.90

19. A pharmacist combines ingredients x, y and z in a ratio of 1:2:7 to produce cough medicine. How many ounces of the second ingredient, ingredient y, is needed to make a 12 ounce bottle of the medicine?

a. 2.4 c. 1.2

b. 8.4 d. 3.6

20. Agency Y served 187,565 people in 1981. If the agency served 210,515 people in 1982, this reflected an increase of:

a. 19.10% c. 12.2%

b. 15.6% d. 10.9%

21. The number of people attending a weekly training program in the month of January averaged 116 people. If there were 105 people attending the first week, 106 the second, and 125 the third, how many people attended the fourth week?

a. 118 c. 130

b. 128 d. 124

22. It takes 16 typists 11 days to complete a project. How long would it take 10 typists, if they worked at the same rate, to complete the same project?

a. 17.6 days c. 6.9 days

b. 6.8 days d. 18.4 days

23. If the sum of two numbers is 280, and their ratio is 7:3, then the smaller number is:

a. 28 c. 56

b. 84 d. 196

24. The population of Metropolis county in 1982 is 130% of its population in 1972. The population in 1972 was 145,000. What was the population in 1982?

a. 196,425 c. 111,539

b. 174,612 d. 188,500

25. A car travels 50 miles an hour, and a plane travels 10 miles a minute. How far will the car travel when the plane travels 500 miles?

a. 50.4 miles c. 41.6 miles

b. 37.5 miles d. 39.7 miles

26. In a university with 2000 students, the student-faculty ratio is 16:1. If 18% of the faculty have completed some of their own study at the university, approximately how many have not?

a. 119 c. 23

b. 127 d. 103

27. A discount house advertises that they sell all merchandise at cost plus 10%. If Jane buys a TV set for $300, approximately what is the store’s profit?

a. $30.00 c. $27.27

b. $27.00 d. $32.26

28. From 6 p.m. until midnight, the temperature dropped at a constant rate. From midnight until 1 a.m., it dropped 8°. If at 6 p.m., the temperature was 54° and by 1 a.m., it was 37°, what was the temperature at 10 p.m.?

a. 46° c. 45°

b. 48° d. 49°

29. One eighth of a half gallon carton of ice cream has been eaten. The remainder is divided among three people. Approximately what percentage of a gallon does each person get?

a. 14.6% c. 29.2%

b. 11.3% d. 18.1%

30. On a promotional exam a woman scored 143 on a scale of 0-160. Her score converted to a scale of 0-100 is approximately:

a. 89 c. 91

b. 70 d. 84

31. A woman paid a tax of $88.00 on property assessed at $28,000. Her neighbor, assessed at the same rate, paid a tax of $110. What was the assessed value of the neighbor’s house?

a. $22,400 c. $35,000

b. $32,400 d. $31,000

32. If Janet can build 22 tables in 14 days, and Anne can build 22 tables in 16 days, approxi- mately how long will it take them to build 22 tables together?

a. 9.5 days c. 15 days

b. 7.5 days d. 8 days

33. Cynthia loaned $35 to Mary. But Cynthia borrowed $14 from Jean, and $16 from Emily. Emily owes $17 to Jean and $9 to Mary. One day they got together to settle their accounts. Who left with $10 less than she came with?

a. Cynthia c. Mary

b. Jean d. Emily

34. How many square tiles, each 12 inches on a side, will Ozzie need to cover a floor that is 11 feet wide and 18 feet long?

a. 99 c. 163

b. 150 d. 198

35. A car has depreciated to 72% of its original cost. If the car is presently valued at $3245, approxi- mately what was its original cost?

a. $5219 c. $4507

b. $5582 d. $2336

36. The sales tax on a typewriter is $13.41 and the sales tax rate is 4%. The purchase price, before the tax was added, was:

a. $335.25 c. $279.10

b. $536.40 d. $317.50

37. What is the interest on $600 at 8% for 30 days?

a. $4 c. $7.50

b. $11.52 d. $4.80

38. A garden is 30 feet by 40 feet. A fence is built around the garden, at a cost of $1.75 per foot of fencing. What was the cost of the fencing?

a. $133.33 c. $210

b. $245 d. $122.50

39. A tax analyst earns four times as much in April as in each of the other months. What part of her entire year’s earnings does she earn in April?

a. 4/11 c. 4/15

b. 1/3 d. 4/13

40. A train travels 70 miles when a bus travels 50 miles. How many miles will the train travel when the bus travels 60 miles?

a. 40 c. 90

b. 78 d. 84

**ANSWER KEY**

**ARITHMETIC REASONING**

1. c 21. b

2. c 22. a

3. d 23. b

4. a 24. d

5. b 25. c

6. b 26. d

7. d 27. c

8. c 28. b

9. a 29. a

10. d 30. a

11. b 31. c

12. a 32. b

13. c 33. d

14. a 34. d

15. d 35. c

16. b 36. a

17. c 37. a

18. a 38. b

19. a 39. c

20. c 40. d

1.3 Analytical Reasoning

Analytical reasoning refers to the ability to look at information, be it [qualitative](https://en.wikipedia.org/wiki/Qualitative_property) or quantitative in nature, and discern patterns within the information. Analytical reasoning involves [deductive reasoning](https://en.wikipedia.org/wiki/Deductive_reasoning) with no specialised knowledge, such as: comprehending the basic structure of a set of relationships; recognizing logically equivalent statements; and inferring what could be true or must be true from given facts and rules. Analytical reasoning is axiomatic in that its truth is self-evident. In contrast, [synthetic reasoning](https://en.wikipedia.org/wiki/Analytic%E2%80%93synthetic_distinction) requires that we include empirical observations, which are always open to doubt.

Directions: Each group of questions in this section is based on a set of conditions. In answering some of the questions, it may be useful to draw a rough diagram. Choose the response that most accurately and completely answers the question and blacken the corresponding space on your answer sheet.

**Passage for Question 1**

A medical clinic has a staff of five doctors—Drs. Albert, Burns, Calogero, Defeo, and Evans. The national medical society sponsors exactly five conferences, which the clinic’s doctors attend, subject to the following constraints: If Dr. Albert attends a conference, then Dr. Defeo does not attend it. If Dr. Burns attends a conference, then either Dr. Calogero or Dr. Defeo, but not both, attends it. If Dr. Calogero attends a conference, then Dr. Evans does not attend it. If Dr. Evans attends a conference, then either Dr. Albert or Dr. Burns, but not both, attends it.

**Question 1**

If Dr. Burns attends one of the conferences, then which one of the following could be a complete and accurate list of the other members of the clinic who also attend that conference?

(A) Drs. Albert and Defeo (B) Drs. Albert and Evans (C) Drs. Calogero and Defeo (D) Dr. Defeo (E) Dr. Evans

**Explanation for Question 1**

This question requires you to determine, from the conditions given, which doctors can attend the same conferences. The question tells us that “Doctor Burns attends one of the conferences,” and we are asked to choose the response that could be a list of all and only those doctors who attend the conference with Dr. Burns. Since we are asked what could be a “complete and accurate list” [emphasis added] of those doctors who attend the conference with Dr. Burns, we can eliminate as incorrect those responses which either are inaccurate (that is, cannot be true), or incomplete (that is, do not include everyone who must accompany one or more of the doctors going to the conference). This can be determined easily without the use of a diagram.

Response (A) states that, along with Dr. Burns, Drs. Albert and Defeo also attend the conference. But the first condition tells us that “if Dr. Albert attends a conference, then Dr. Defeo does not attend it.” So, Drs. Burns, Albert, and Defeo cannot all attend the same conference. Response (A), then, is incorrect.

Response (B) is incorrect for a similar reason. The fourth condition tells us what must be true if Dr. Evans attends a conference, namely, that “either Dr. Albert or Dr. Burns, but not both, attends it.” Since we know that Dr. Burns attends the conference, we know that it cannot be true that both Drs. Albert and Evans also attend that conference.

Response (C) is also incorrect. The second condition tells us what must be true if Dr. Burns attends a conference. Since we know that Dr. Burns does attend the conference, we also know that “either Dr. Calogero or Dr. Defeo, but not both, attends it.”

Responses (D) and (E) must be evaluated slightly differently. No condition rules out Dr. Burns’s and Dr. Defeo’s going to the same conference—response (D)—and no condition forbids Dr. Evans’s going with Dr. Burns to a conference—response (E). But recall that the question asks for what could be a “complete and accurate list” of the doctors who attend the conference with Dr. Burns. We know from the second condition that at least one other person must accompany Dr. Burns, and that among those who accompany Dr. Burns are either Dr. Calogero or else Dr. Defeo. Since the conditions do not require anyone to accompany Dr. Defeo, it is possible that Dr. Defeo is the only person to accompany Dr. Burns. Thus, response (D) is an accurate response, in that it is possible that Drs. Burns and Defeo attend the same conference, and it is a complete response, in that Drs. Burns and Defeo could be the only doctors of the five to attend the conference. So response (D) is correct.

Response (E) is incorrect because we know that if Dr. Burns goes, someone other than Dr. Evans must also go. Response (E) then is incomplete. It fails to list at least one doctor whom we know must also accompany Dr. Burns.

This question is classified as “moderately difficult.”

**Passage for Questions 2 and 3**

Seven piano students—T, U, V, W, X, Y, and Z—are to give a recital, and their instructor is deciding the order in which they will perform. Each student will play exactly one piece, a piano solo. In deciding the order of performance, the instructor must observe the following restrictions: X cannot play first or second. W cannot play until X has played. Neither T nor Y can play seventh. Either Y or Z must play immediately after W plays. V must play either immediately after or immediately before U plays.

**Question 2**

If V plays first, which one of the following must be true?

(A) T plays sixth. (B) X plays third. (C) Z plays seventh. (D) T plays immediately after Y. (E) W plays immediately after X.

**Explanation for Question 2**

This question deals with an ordering relationship defined by a set of conditions as to when the seven piano students will perform. As an aid in visualizing this problem we can draw a simple diagram that shows the seven recital slots arranged in order from left to right. Student V is shown in the first slot, as specified by the condition that “V plays first”:

We can immediately fill in one of the empty slots in the diagram. The condition that “V must play either immediately after or immediately before U plays” tells us that U must occupy the second slot in the recital schedule. This is shown below:

Since the question asks us what must be true, we can eliminate incorrect responses by showing that they could be false without violating the conditions. Response (A) is incorrect because the statement that “T plays sixth” is not necessarily true—we can place T in one of the slots other than sixth and still meet all the conditions of the problem. One such recital schedule, with T playing third, is shown in the diagram below:

We can develop this schedule as follows. With V, U, and T in the first three positions, there are four positions left for W, X, Y, and Z.

W must come after X—because of the condition that “W cannot play until X has played”—so if we put X fourth and W fifth, this condition will be met.

This leaves two possible slots for Y and Z. Y cannot play seventh because of the condition that “Neither T nor Y can play seventh,” so we will place Y sixth and Z seventh.

A check will verify that this schedule meets the given conditions of the problem, including the one that “Either Y or Z must play immediately after W plays.”

The schedule shown in the diagram also demonstrates that response (B) is incorrect. In it, X plays fourth, so it is not correct that the statement, “X plays third,” must be true.

Response (C), “Z plays seventh,” is the credited response. We can show this by demonstrating that:

all the conditions can be met with Z in the seventh slot, and

some of the conditions would be violated with Z in any slot other than seventh.

To demonstrate that Z can play seventh, we can refer to the schedule that was developed for the discussion of response (A), above. In it, Z plays seventh, and all the conditions in the problem are met.

To demonstrate that Z cannot play in a slot other than seventh, we can attempt to find another student to play seventh. We already know that neither U nor V can play seventh. Hence, there are four remaining players: T, W, X, and Y. However, a review of the given conditions shows that none of those players can play seventh:

The third condition states that “Neither T nor Y can play seventh.”

W can’t play seventh, because there must be a slot following W’s in order to meet the condition, “Either Y or Z must play immediately after W plays.” If W plays seventh, then there is no such slot left for Y or Z.

For a similar reason X can’t play seventh, because there must be a slot following X’s in order to meet the condition, “W cannot play until X has played.”

Since Z can play seventh and no other player can, then the statement that Z must play seventh is correct and (C) is the credited response.

Response (D) is incorrect because it is not necessarily true that “T plays immediately after Y.” In our discussion of response (A), we developed a schedule in which T plays third and Y plays sixth, yet all conditions are satisfied.

Response (E) is incorrect because, as shown in the diagram below, it is not necessarily true that “W plays immediately after X.” This schedule is obtained by simply reversing the order of players W and Y in the schedule we developed in the analysis of response (A).

A review will show that all of the given conditions are met by this schedule.

This question is classified as “difficult.”

**Question 3**

If U plays third, what is the latest position in which Y can play?

(A) first (B) second (C) fifth (D) sixth (E) seventh

**Explanation for Question 3**

This question involves the same original conditions as the previous problem, but it begins with a different assumption: “U plays third.” The test taker must determine what effect this assumption would have on the possible positions in which Y can appear in the recital schedule.

The correct response is (D), because student Y can play as late as sixth under the given constraint that “U plays third.” The diagram below shows a recital order that meets all the given conditions and has Y performing in the sixth position.

One strategy for arriving at this solution is to work backward to see which position is the latest in which we can place Y and at the same time produce a recital schedule that meets all the given conditions.

Using that approach, we immediately see that Y cannot play as late as seventh, because of the condition that “Neither T nor Y can play seventh.” Backing up and placing Y sixth, we can begin to fill in the schedule, as follows:

This schedule has five empty slots, into which we must fit players T, V, W, X, and Z. The following is a series of reasoning steps that can be used:

From our analysis of the previous question, we know that players T, W, and X cannot play seventh, but that Z can, so we can tentatively place Z in the seventh slot.

We also know that “Either Y or Z must play immediately after W plays.” If we place W in the fifth slot, this condition will be met.

By placing V in the second slot, we can meet the condition that “V must play either immediately after or immediately before U plays.”

We must place the remaining two players, T and X, in the two remaining slots, the first and the fourth. Because the first condition states that “X cannot play first … ,” we will place X in the fourth slot and T in the first. These positions will meet the conditions that apply to T and X: T will avoid playing seventh and X will play before W.

Since Y can play as late as sixth, response (D) is the correct solution.

This question is classified as “moderately difficult.”

**Passage for Questions 4 and 5**

On a particular Saturday, a student will perform six activities— grocery shopping, hedge trimming, jogging, kitchen cleaning, laundry, and motorbike servicing. Each activity will be performed once, one at a time. The order in which the activities are performed is subject to the following conditions: Grocery shopping has to be immediately after hedge trimming. Kitchen cleaning has to be earlier than grocery shopping. Motorbike servicing has to be earlier than laundry. Motorbike servicing has to be either immediately before or immediately after jogging.

**Question 4**

If laundry is earlier than kitchen cleaning, then hedge trimming must be

(A) fifth (B) fourth (C) third (D) second (E) first

**Explanation for Question 4**

This problem is concerned with determining the order in which six activities will be performed. As with many questions involving relative ordering or ranking, it is likely that you will find it useful to diagram the various relationships given in the passage.

The first condition in the passage tells us that grocery shopping has to be immediately after hedge trimming, which we can abbreviate as follows:

1. HG

The second condition tells us that kitchen cleaning has to be earlier than grocery shopping, which we can abbreviate as follows, where “…” is used to represent “earlier than” (which means any time before, including immediately before):

2. K…G

The third condition tells us that motorbike servicing has to be earlier than laundry, and the fourth condition tells us that motorbike servicing has to be either immediately before or immediately after jogging. These conditions can be abbreviated as follows, where the / symbol is used to represent “or”:

3. M…L

4. MJ/JM

Notice that the information specified in these four conditions can be collapsed into two ordering statements:

I. K…HG (first and second conditions)

II. MJ/JM…L (third and fourth conditions)

Question 7 introduces the new supposition “laundry is earlier than kitchen cleaning”:

L…K

This new supposition works to further collapse the ordering statements in I and II to the single statement below; that is, if L must be earlier than K, then we know that the activities must be ordered like this:

MJ/JM…L…K…HG

So, with the addition of the new supposition, there are exactly two possible orderings of the six activities, differing only with respect to whether motorbike servicing is immediately before or immediately after jogging:

1 2 3 4 5 6 M J L K H G J M L K H G

Question 7 asks what position hedge trimming must be in, given the new supposition. What we see here is that hedge trimming must be the fifth activity performed, and so answer choice (A) is correct.

This question is classified as “easy.”

**Question 5**

Which one of the following, if substituted for the condition that motorbike servicing has to be earlier than laundry, would have the same effect in determining the order of the student’s activities?

(A) Laundry has to be one of the last three activities. (B) Laundry has to be either immediately before or immediately after jogging. (C) Jogging has to be earlier than laundry. (D) Laundry has to be earlier than hedge trimming. (E) Laundry has to be earlier than jogging.

Explanation for Question 5

This question asks you to select the condition which, if substituted for the third condition in the passage (repeated below), would have the same effect as the original condition.

Third condition: Motorbike servicing has to be earlier than laundry.

In this case, you can deduce that the correct answer choice is (C):

(C) Jogging has to be earlier than laundry.

The fourth condition in the passage tells you that motorbike servicing has to be either immediately before or immediately after jogging. That is, M and J must be ordered as a block, either MJ or JM, with respect to the other four activities. Thus, if, as the original third condition states, M has to be earlier than L, then we know that J must also be earlier than L. Conversely, if, as the new condition in answer choice (C) states, J has to be earlier than L, then we know that M must also be earlier than L. In short, the third condition and answer choice (C) have exactly the same effect. Therefore, answer choice (C) is correct.

Another way to approach this kind of question is to attempt to eliminate all of the incorrect answer choices. Under this approach, you want to rule out any answer choice that does either of the following:

1. rules out outcomes that the original condition allows
2. allows outcomes that the original condition rules out

Let’s see how this approach would enable us to eliminate answer choices (A), (B), (D), and (E).

Consider the condition presented in answer choice (A):

(A) Laundry has to be one of the last three activities.

We can first ask whether this condition would rule out outcomes that the original third condition allows. To answer this question, we must simply determine whether there is an outcome allowed by the original third condition along with the other conditions in which laundry is one of the first three activities. Here is such an outcome:

1 2 3 4 5 6 M J L K H G

Because the original third condition allows this outcome, but the condition in answer choice (A) does not, answer choice (A) cannot be correct.

Consider answer choice (B):

(B) Laundry has to be either immediately before or immediately after jogging.

Again, we want to first determine whether this new condition would rule out outcomes that the original third condition allows. To answer this question, we must simply determine whether there is at least one outcome allowed by the original third condition along with the other conditions in which laundry is neither immediately before nor immediately after jogging. Here is one such outcome:

1 2 3 4 5 6 K H G J M L

This outcome, although allowed by the original third condition, would be ruled out by the alternative condition given in answer choice (B). Thus, answer choice (B) cannot be correct.

Next consider answer choice (D):

(D) Laundry has to be earlier than hedge trimming.

Again, we want to first determine whether this new condition would rule out outcomes that the original third condition allows. To answer this question, we must simply determine whether there is at least one outcome allowed by the original third condition along with the other conditions in which laundry is not earlier than hedge trimming. One such outcome was given immediately above: since L is not earlier than H in this outcome, it would be ruled out by the condition in answer choice (D). So, answer choice (D) rules out an outcome that the original third condition allows, and therefore (D) cannot be the correct answer choice.

Finally, consider answer choice (E):

(E) Laundry has to be earlier than jogging.

Again, we want to first determine whether have this new condition would rule out outcomes that are allowed when the original third condition is in place. To answer this question, we must simply determine whether there is at least one outcome allowed by the original third condition along with the other conditions in which laundry is not earlier than jogging. One such outcome was given above: since L is not earlier than J in this outcome, it would be ruled out by the condition presented in answer choice (E). So, answer choice (E) rules out an outcome that the original third condition allows, and therefore (E) cannot be the correct answer choice.

In sum, answer choices (A), (B), (D), and (E) can all be eliminated because in each case the condition is one that rules out outcomes that the original condition allows. For this particular question, there was no need to consider whether any of the options could be eliminated because they allowed outcomes that the original condition ruled out.

This question is classified as “moderately difficult.”

1.4 Fundamentals of algorithms and flowcharts with examples

**Algorithm:**

An algorithm is defined as unambiguous, step by step procedure (instructions) to solve a given problem in finite number of steps by accepting a set of inputs and producing the desired output. After producing the result, the algorithm should terminate. An algorithm is ‘an effective procedure for solving a problem in a finite number of steps. Usually, Algorithms are written in simple English like statements along with simple mathematical expressions.

**Algorithm to add 2 numbers:**

* Step 1:Start
* Step 2: [Read 2 values a and b to be added]

Read a,b.

* Step 3: [Find the sum of 2 numbers]

sum=a+b.

* Step 4: [Output the result]

Print sum.

* Step 5: [End of Algorithm]

Stop

**Algorithm to find Area of Rectangle:**

* Step 1:Start
* Step 2: [Read the values of length and breadth]

Read length, breadth.

* Step 3: [Find the area of rectangle]

Area=length\*breadth.

* Step 4: [Output the result]

Print area.

* Step 5: [End of Algorithm]

Stop

**Algorithm to find swap 2 numbers:**

* Step 1:Start
* Step 2: [Read the values A and B]

Read A,B

* Step 3: [Find A AND B]

C=A

A=B

B=C

* Step 4: [Output the result]

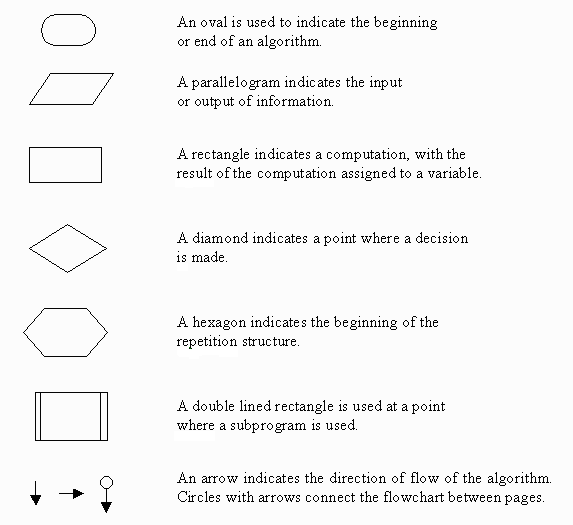
Print A AND B.

* Step 5: [End of Algorithm]

Stop

**Flowchart:**

* A flowchart is a pictorial representation of an algorithm. That is flowchart consists of sequence of instructions that are carried out in an algorithm. All the steps are drawn in the form of different shapes of boxes, circle and connecting arrows. Flowcharts are mainly used to help programmer to understand the logic/the flow of the program.
* The various types of geometric shapes, arrows and symbols used while drawing the flow chart are called flowchart symbols. The symbols used and the meaning associated with each symbol are shown below in fig 1.

[](https://www.google.co.in/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=0CAcQjRxqFQoTCIDy_bCqgscCFYwKjgodOn8K5Q&url=http://jabroo.blogspot.com/2011/02/flowchart-and-algorithm.html&ei=GdK5VcCBJoyVuAS6_qmoDg&bvm=bv.99028883,d.c2E&psig=AFQjCNElvYbQximTWG2MFJgpEFD4Q1fXlQ&ust=1438327692132145)

**Fig 1: Flow chart symbols**

**Flow chart to find the area of rectangle:**

**START**

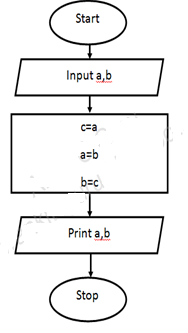
**Read Length and Breadth**

**Area=length\*breadth**

**Print Area**

**STOP**

**Flowchart for Swapping contents of 2 variables:**

****

**Algorithm & Flowchart to find the Largest of given three numbers:**

**Algorithm :**

Step 1: Start

Step 2: Read three numbers A, B, C

Step 3: Compare A with B

Step 4: If A is larger compare it with C

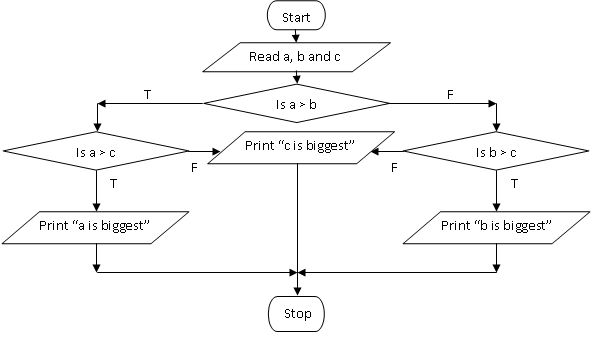
Step 5: If A is larger than C then A is the largest otherwise C is the largest.

Step 6: If A is smaller than or equal to B in the first step then B is compared with C.

Step 7: If B is larger than C then B is the largest number otherwise C is the largest number.

Step 8: Stop

**Flowchart :**



**Algorithm &Flowchart for solving a given quadratic equation:**

Algorithm to find theRoots of a Quadratic equation

**Step 1:** Start

**Step 2:** Read the values of non-zero coefficients a,b and c.

**Step 3:** Compute the value of discriminant (disc) which is equal to (b\*b-4\*a\*c).

**Step 4:** Check if disc is equal to 0. If true, then go toStep 5. Otherwise, go to Step 6.

**Step 5:**Compute the roots.

root1 = (-b)/ (2\*a)

root2=root1

Output the values of roots, root1 and root2.

Go to Step 9.

**Step 6:** Check if disc is greater than zero or not. If true, then go to Step 7.Otherwise, go to Step 8.

**Step 7:**Compute the real and distinct roots,

root1 = (-b+sqrt(disc))/(2\*a)

root2 = (-b-sqrt(disc))/(2\*a)

Output the values of roots, root1 and root2.

Go to Step 9.

**Step 8:**Compute the complex and distinct roots.

Compute the real part, r\_part = (-b)/(2\*a)

Compute the imaginary part, i\_part = sqrt(-disc)/(2\*a)

Output roots as root1 = r\_part + i\_part

root2 = r\_part – i\_part

**Step 9:** Stop

**START**

Read a, b, c

disc=b\*b-4\*a\*c

If

disc =0?

root1 = -b/(2\*a)root2 = root1

root1 = (-b+sqrt(disc))/(2\*a)

root2 = (-b-sqrt(disc))/(2\*a)

r\_part = -b/(2\*a)

i\_part = sqrt(-(disc))/(2\*a)

Output root1,root2

With suitable message

Output root1,root2

Withsuitablemessage

Output root1 as

r\_part + i\_part,

root2 as

r\_part - i\_ part

**STOP**

If

disc> 0?

TRUE

TRUE

FALSE

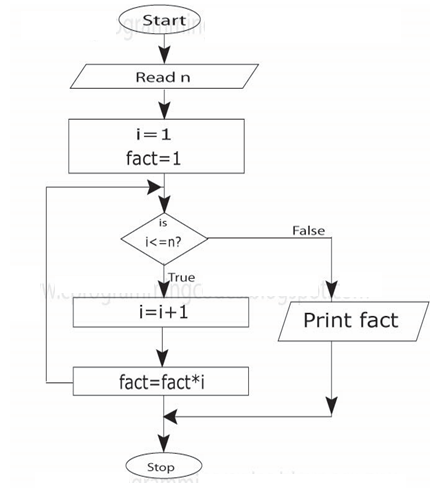
FALSE

**Algorithm & flowchart to find the Factorial of a given integer:**

**Algorithm :**

Step 1. Start  
Step 2. Read the number n  
Step 3. [Initialize]  
        i=1, fact=1  
Step 4. Repeat step 4 through 6 until i=n  
Step 5. fact=fact\*i  
Step 6. i=i+1  
Step 7. Print fact  
Step 8. Stop  
[process finish of calculate the factorial value of a number]

**Flowchart:**

****

2. INTRODUCTION TO C PROGRAMMING

**Programming** is the process of taking an algorithm and encoding it into a notation, a programming language, so that it can be executed by a computer. Although many programming languages and many different types of computers exist, the important first step is the need to have the solution. Without an algorithm there can be no program.

Computer science is not the study of programming. Programming, however, is an important part of what a computer scientist does. Programming is often the way that we create a representation for our solutions. Therefore, this language representation and the process of creating it becomes a fundamental part of the discipline.

Algorithms describe the solution to a problem in terms of the data needed to represent the problem instance and the set of steps necessary to produce the intended result. Programming languages must provide a notational way to represent both the process and the data. To this end, languages provide control constructs and data types.

Control constructs allow algorithmic steps to be represented in a convenient yet unambiguous way. At a minimum, algorithms require constructs that perform sequential processing, selection for decision-making, and iteration for repetitive control. As long as the language provides these basic statements, it can be used for algorithm representation.

All data items in the computer are represented as strings of binary digits. In order to give these strings meaning, we need to have **data types**. Data types provide an interpretation for this binary data so that we can think about the data in terms that make sense with respect to the problem being solved. These low-level, built-in data types (sometimes called the primitive data types) provide the building blocks for algorithm development.

2.1 Programming paradigms:

A programming paradigm refers to the way of problem solving that includes a set of methodologies, theories, practices and standards. Programming languages can be classified in various ways. According to the extent of translation that is required to generate the machine instructions from a program, programming languages can be classified into low-level or high-level languages. Both assembly language and machine language are considered as low-level languages.

Low-level languages are closer to the native language of the computer as program written in machine language does not require translation for a processor to execute them. Assembly language is also considered as a low-level language since each assembly language instruction accomplishes only a single operation and the coding for a problem is at the individual instruction level. On the other hand, high-level programming languages provide a high level of abstraction from the actual machine hardware.

High-level languages can further be characterized is by programming paradigm as shown in figure 1.

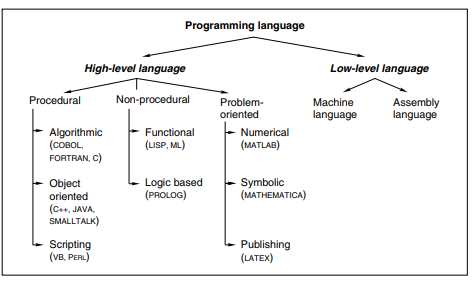


Figure 1: Classifiation of languages

The highlevel programming languages may also be categorized into three groups—procedural, non-procedural, and problem oriented.

Procedural programming languages In procedural programming, a program is conceived as a set of logically related instructions to be executed in order. Each program can be divided into small self-contained program segment, each of which performs a particular task and be re-used in the program as and when required without repeated explicit coding corresponding to the segment.

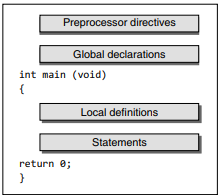
These sections of code are known as procedures or subroutines or functions. It also makes it easier for programmers to understand and maintain program structure. There are mainly three classes of procedural programming languages- Algorithmic, Object oriented and Scripting.

Non Procedural Languages These languages are called rule based languages or logic programming languages. A logic program is expressed as a set of atomic sentences, known as facts, and horn clauses, such as if-then rules. A query is then posed. Then the execution of the program begins and the system tries to find out if the answer to the query is true or false for the given facts and rules.

Problem-oriented languages These languages provide readymade procedures or functions which are pre-programmed. The user has to write the statements in terms of those pre-written functions. MATLAB is a very popular language among scientists and engineers to solve a wide class of problems in digital signal processing, control systems, modelling of systems described by differential equations, matrix computations.

2.2 Structure of C programs

The general structure of a C program is depicted in the figure 2.



**Figure 2. Structure of C program.**

*Preprocessor directives* tell the preprocessor to look for special code libraries, make substitutions in the code, and in other ways prepare the code for translation into machine language.

*Global declared* variables are visible to all parts of the program.

*int main( )*- indicates the beginning of the program.

{ - Opening braces- Indicates the beginning of the program.

All functions including main() can be divided into two sections—local definition and statements.

*Local definitions*- Describes the data that will be used.

*Statement section* consists of the instructions that cause the computer to do something.

*Note:*

*Declaration means describing the type of a data object to the compiler but not allocating any space for it.*

*Definition means declaration of a data object and also allocating space to hold the data object.*

}- Closing braces- Indicates end of the program.

2.3 The process of compiling and running the C program

There are mainly three steps in developing a program in C:

1. Writing the C program 2. Compiling the program 3. Executing the program.

Figure 3 shows the steps of compiling and executing C Program.

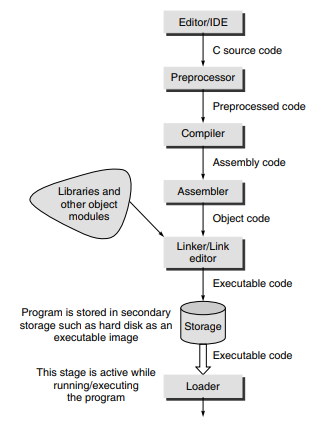


Figure 3. Typical steps for entering, compiling and executing C programs

Writing a C Program- This involves writing a new program code or editing an existing source program using a text editor or an IDE and saving it with .c extension.

Compiling a C Program- Compiling involves preprocessing, compilation, assembly, and linking.

*Preprocessing* It is the first phase of C compilation. It processes include-files, conditional compilation instructions, and macros. . It is usually hidden from the programmer because it is run automatically by the compiler.

*Compilation* It is the second step of the compiling process. It takes the output of the preprocessor and the source code, and generates assembler source code. The compiler examines each program statement contained in the source program and checks it to ensure that it conforms to the syntax and semantics of the language. If mistakes are discovered by the compiler during this phase, they are reported to the user. The errors then have to be corrected in the source program (with the use of an editor), and the program has to be recompiled.

*Assembly* It is the third stage of compilation. It takes the assembly source code and produces an assembly listing with offsets. The assembler output is stored in an object file. On most systems, the assembler is executed automatically as part of the compilation process. The assembler takes each assembly language statement and converts it into a binary format known as object code, which is then written into another file on the system. This file typically has the same name as the source file under UNIX, with the last letter an ‘o’ (for object) instead of a ‘c’. Under Windows, the suffix letters “obj” typically replace the “c” in the filename.

*Linking* It is the final stage of compilation. After the program has been translated into object code, it is ready to be linked. The purpose of the linking phase is to get the program into a final form for execution on the computer.

\*\*The process of compiling and linking a program is often called building.

Execution of C program The final linked file, which is in an executable object code format, is stored in another file on the system ready to be run or executed. Under UNIX, this file is called a.out by default. Under Windows, the executable file usually has the same name as the source file, with the .c extension replaced by an exe extension

2.4 Character set

*character:-* It denotes any alphabet, digit or special symbol used to represent information.

*Use*:- These characters can be combined to form variables. C uses constants, variables, operators, keywords and expressions as building blocks to form a basic C program.

*Character set*:- The character set is the fundamental raw material of any language and they are used to represent information. Like natural languages, computer language will also have well defined character set, which is useful to build the programs.

The characters in C are grouped into the following two categories:

1. Source character set

|  |
| --- |
| **~** tilde                            %         percent sign               |           vertical bar         @        at symbol               +          plus sign                    <          less than |
|  |
| \_          underscore                -           minus sign                 >          greater than       ^          caret                        #          number sign              =          equal to |
|  |
| &         ampersand                 $          dollar sign                 /           slash                  (           left parenthesis      \*          asterisk                      \           back slash |
|  |
| )           right parenthesis       ′           apostrophe                :           colon                  [           left bracket               "          quotation mark         ;           semicolon |
|  |
| ]           right bracket              !           exclamation mark     ,           comma                { left flower brace               ?            Question mark       .           dot operator |
|  |
| }            right flower brace |

a. Alphabets

b. Digits

c. Special Characters

d. White Spaces

2. Execution character set

a. Escape Sequence

Source character set

ALPHABETS

Uppercase letters A-Z

Lowercase letters a-z

DIGITS 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

**Special characters:**

Whitespace characters:

|  |
| --- |
| \b         blank space               \t          horizontal tab                       \v         vertical tab             \r         carriage return          \f         form feed                   \n         new line |
|  |
| \\          Back slash                 \’         Single quote                          \"         Double quote       \?         Question mark          \0         Null                            \a         Alarm (bell) |

Execution Character Set:

Certain ASCII characters are unprintable, which means they are not displayed on the screen or printer. Those characters perform other functions aside from displaying text. Examples are backspacing, moving to a newline, or ringing a bell.

They are used in output statements. Escape sequence usually consists of a backslash and a letter or a combination of digits. An escape sequence is considered as a single character but a valid character constant.

These are employed at the time of execution of the program. Execution characters set are always represented by a backslash (\) followed by a character. Note that each one of character constants represents one character, although they consist of two characters. These characters combinations are called as escape sequence.

**Backslash character constants**

|  |
| --- |
| Character                      ASCII  value    Escape Sequence             Result |
| Null                                         000                   \0                                    Null |
|  |
| Alarm (bell)                            007                  \a                                    Beep Sound |
|  |
| Back space                            008                  \b                                   Moves previous position |
|  |
| Horizontal tab                        009                  \t                                    Moves  next horizontal tab |
|  |
| New line                                 010                  \n                                   Moves  next Line |
|  |
| Vertical tab                             011                  \v                                   Moves  next vertical tab |
|  |
| Form feed                              012                  \f                                    Moves initial position of next page |
|  |
| Carriage return                     013                  \r                                    Moves beginning of the line |
|  |
| Double quote                       034                   \"                                    Present Double quotes |
|  |
| Single quote                         039                   \'                                    Present Apostrophe |
|  |
| Question mark                     063                   \?                                  Present Question Mark |
|  |
| Back slash                            092                   \\                                   Present back slash |
|  |
| Octal number                       \000 |
|  |
| Hexadecimal number        \x |

2.5 Features of C Programming Language:

* Procedural Language
* Fast and Efficient
* Modularity
* Statically Type
* General Purpose Language
* Rich set of built in Operators
* Libraries with rich Functions
* Middle Level Language
* Portability
* Easy to Extend

2.6 C tokens

It is the basic **building block** of a C-Program. There are five classes of tokens: identifiers,

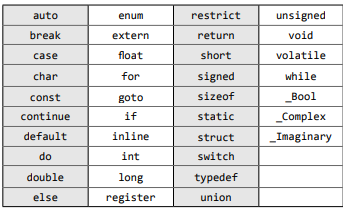
reserved words, operators, separators, and constants.

1. *Identifier* It is a sequence of characters invented by the programmer to identify or name a specific object, and the nameis formed by a sequence of letters, digits, and underscores.
2. *Keywords* These are explicitly reserved words that have a strict meaning as individual tokens to the compiler. They cannot be redefined or used in other contexts. Use of variable names with the same name as any of the keywords will cause a compiler error.
3. *Operators* These are tokens used to indicate an action to be taken (usually arithmetic operations, logical operations, bit operations, and assignment operations). Operators can be simple operators (a single character token) or compound operators (two or more character tokens).
4. *Separators* These are tokens used to separate other tokens. Two common kinds of separators are indicators of an end of an instruction and separators used for grouping.
5. *Constant* It is an entity that does not change.

2.7 Keywords and Identifiers

**Keywords –** Keywords are the vocabulary of C. Because they are special to C, one cannot use them for variable names. There are 32 words defined as keywords in C. These have predefined uses and cannot be used for any other purpose in a C program. They are used by the compiler to compile the program. They are always written in lowercase letters.

Full set of keywords upto C99



**Identifiers** - In C, variables, arrays, functions, and labels are named. Describing them may help to learn something about the character of the language since they are elements that C permits the programmer to define and manipulate.

Some rules must be kept in mind when naming identifiers. These are stated as follows:

1. The first character must be an alphabetic character (lower-case or capital letters) or an underscore ‘\_’.

2. All characters must be alphabetic characters, digits, or underscores.

3. The first 31 characters of the identifier are significant. Identifiers that share the same first 31 characters may be indistinguishable from each other.

4. A keyword cannot be duplicated by an identifier. A keyword is word which has special meaning in C.

Some examples of proper identifiers are *employee\_number, box\_4\_weight, monthly\_pay, interest\_per\_annum, job\_number, and tool\_4.*

Some examples of incorrect identifiers are *230\_item, #pulse\_rate, total~amount, /profit margin, and ~cost\_per\_ item.*

2.8 Constants

The constants refer to fixed values that the program may not alter during its execution. These fixed values are also called **literals**.

A constant is an explicit data value written by the programmer. Thus, it is a value known to the compiler at compiling time. The compiler may deal with this value in any of several ways, depending on the type of constant and its context. For example, the binary equivalent of the constant may be inserted directly into the output code stream. The value of the constant may be stored in a special data area in memory. The compiler may decide to use the constant’s value for its own immediate purpose, e.g., to determine how much storage it should allocate to a data array

Constants can be of any of the basic data types like *an integer constant, a floating constant, a character constant, or a string literal*.

*Integer Constant*- An integer constant consists of a sequence of digits. It is normally interpreted as a decimal value. Thus, 1, 25, and 23456 are all decimal integer constants. A literal integer (e.g., 1984) is always assumed to be of type int, unless it has an ‘L’ or ‘l’ suffix, in which case it is treated as a long. Also, a literal integer can be specified to be unsigned using the suffix U or u. For example,

1984L 1984l 1984U 1984u 1984LU 1984ul

*Floating point constant*- A floating-point constant consists of an integer part, a decimal point, a fractional part, and an exponent field containing an e or E followed by an integer. Both integer and fractional parts are digit sequences. In addition to the decimal notation used so far, literal reals may also be expressed in scientific notation. For example, 0.002164 may be written in scientific notation as 2.164E-3 or 2.164e-3 The letter E (or e) stands for exponent. The scientific notation is interpreted as follows: 2.164E-3 = 2.164 × 10^-3

*Character Constant-* A character constant normally consists of a single character enclosed in single quotes. Thus, for example, ‘b’ and ‘$’ are both character constants. Each takes on the numeric value of its character in the machine’s character set. Unless stated otherwise, it will henceforth be assumed that the ASCII code is used. Thus, for example, writing down the character constant ‘A’ is equivalent to writing down the hex value 41 or the octal value 101. The ‘A’ form is preferable, of course, first, because its meaning is unmistakable, and second, because it is independent of the actual character set of the machine.

*Note: Backslash character Format \(single character) It escapes the regular meaning of the character and gives some special meaning Ex: \n—new line character*

*String Constant-* A string constant is a sequence of characters enclosed in double quotes. Whenever the C compiler encounters a string constant, it stores the character sequence in an available data area in memory. It also records the address of the first character and appends to the stored sequence an additional character, the null character ‘\0’, to mark the end of the string. The length of a character string is the number of characters in it (again, excluding the surrounding double quotes). Thus, the string “messagen” has a length of eight

2.9 Variables

A variable is a name given to a storage area that programs can manipulate. Each variable in C has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.

The name of a variable(called as identifier) can be composed of letters, digits, and the underscore character. The rules discussed for naming the identifiers should be followed for variable names.

2.10 Data types in C

The type or data type of a variable determines the set of values that a variable might take and the set of operations that can be applied to those values. Data types can be broadly classified as shown in figure 4

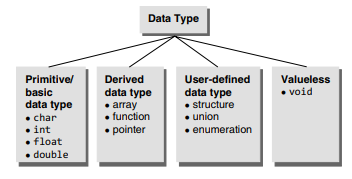


Figure 4. Data types in C

Basic Data Types: C has five basic data types, More complex data types can be built up from these basic types and they are:

1. character—Keyword used is char
2. integer—Keyword used is int
3. floating-point—Keyword used is float
4. double precision floating point—Keyword used is double
5. valueless—Keyword used is void

The following table lists the sizes and ranges of basic data types in C for 16bit and 32 bit machine

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **For 16 bit machine** | | **For 32 bit machine** | |
| *Data Type* | *Size in bits* | *Range* | *Size in bits* | *Range* |
| char | 8 | -128 to 127 | 8 | –128 to 127 |
| int | 16 | –32768 to 32767 | 32 | –2147483648 to 2147483647 |
| float | 32 | 1.17549 × 10–38 to 3.40282 × 1038 | 32 | 1.17549 × 10–38 to 3.40282 × 1038 |
| double | 64 | 2.22507 × 10–308 to 1.79769 × 10 308 | 64 | 2.22507 × 10–308 to 1.79769 × 10 308 |
| void | 8 | valueless | 8 | valueless |

*Type qualifiers*

C has four type specifiers or modifiers and three type qualifiers. The following points should be noted:

(a) Each of these type of modifiers can be applied to the base type int.

(b) The modifiers signed and unsigned can also be applied to the base type char.

(c) In addition, long can be applied to double.

(d) When the base type is omitted from a declaration, int is assumed.

(e) the type void does not have these modifiers.

The specifiers and qualifiers for the data types can be broadly classified into three types:

*Size specifiers—short and long*

*Sign specifiers—signed and unsigned*

*Type qualifiers—const, volatile, and restrict*

*Size specifiers*-The size qualifiers alter the size of the basic data types. There are two such qualifiers that can be used with the data type int; these are short and long. The specifier short, when placed in front of the int declaration, tells the C compiler that the particular variable being declared is used to store fairly small integer values. The long qualifier is also used with the basic data type double.

Size of short int, int and long int is shown below.

|  |  |  |  |
| --- | --- | --- | --- |
| *Data Type* | *16 bit machine* | *32 bit machine* | *64 bit machine* |
| short int | 2 | 2 | 2 |
| int | 2 | 4 | 4 |
| long int | 4 | 4 | 8 |

\*C99 provides two additional integer types: long long int and unsigned long long int. For long long, the C99 standard specified at least 64 bits to support.

*Sign specifiers-* The precise range of values representable by a signed integer type depends not only on the number of bits used in the representation, but also on the encoding techniques. The most common binary encoding technique for integers is called 2’s complement notation in which a signed integer represented with n bits will have a range from (–2n–1) through (2n–1 – 1) encoded in the following fashion:

1. The highest order(left-most) bit (of the word) is the sign bit. If the sign bit is 1, the number is negative; otherwise the number is positive.

2. To negate an integer, complement all bits in the word and then add 1 to the result; thus to form the integer –1, start with 1 (00….0012), complement the bits 11 ….. 1102, and add 1 giving 11….1112 = –1.

3.The maximum negative value, 10…..00002 or –2n–1, has no positive equivalent; negating this value produces the same value

*Unsigned Numbers*- For each signed integer types, there is a corresponding unsigned type that occupies the same amount of storage but has a different integer encoding. The declaration of unsigned int variable ‘liberates’ the sign bit and makes the entire word (including the freed sign bit) available for the storage of non-negative integers.

*Type qualifiers*- control the way variables may be accessed or modified. They specify the variables that will never (const) change and those variables that can change unexpectedly (volatile). Both keywords require that an associated data type be declared for the identifier, for example, const float pi = 3.14156; specifies that the variable pi can never be changed by the program. Any attempt by code within the program to alter the value of pi will result in a compile time error. The volatile keyword indicates that a variable can unexpectedly change because of events outside the control of the program.This is usually used when some variable within the program is linked directly with some hardware component of the system. The restrict type qualifier allows programs to be written so that translators can produce significantly faster executables.

**Declaration of variables**

A variable definition means to tell the compiler where and how much to create the storage for the variable. A variable definition specifies a data type and contains a list of one or more variables of that type as follows:

**type variable\_list;**

Here, **type** must be a valid C data type including char, w\_char, int, float, double, bool or any user-defined object, etc., and **variable\_list** may consist of one or more identifier names separated by commas. Some valid declarations are shown here:

int i, j, k;

char c, ch;

float f, salary;

double d;

The line **int i, j, k;** both declares and defines the variables i, j and k; which instructs the compiler to create variables named i, j and k of type int.

Variables can be initialized (assigned an initial value) in their declaration. The initializer consists of an equal sign followed by a constant expression as follows:

type variable\_name = value;

Some examples are:

extern int d = 3, f = 5; // declaration of d and f.

int d = 3, f = 5; // definition and initializing d and f.

byte z = 22; // definition and initializes z.

char x = 'x'; // the variable x has the value 'x'.

For definition without an initializer: variables with static storage duration are implicitly initialized with NULL (all bytes have the value 0); the initial value of all other variables is undefined.

A variable declaration provides assurance to the compiler that there is one variable existing with the given type and name so that compiler proceed for further compilation without needing complete detail about the variable. A variable declaration has its meaning at the time of compilation only, compiler needs actual variable declaration at the time of linking of the program.

A variable declaration is useful when you are using multiple files and you define your variable in one of the files which will be available at the time of linking of the program. You will use extern keyword to declare a variable at any place. Though you can declare a variable multiple times in your C program but it can be defined only once in a file, a function or a block of code.

2.11 Pre-processor directives:

The C preprocessor modifies a source code file before handing it over to the compiler. You're most likely used to using the preprocessor to include files directly into other files, or #define constants, but the preprocessor can also be used to create "inlined" code using macros expanded at compile time and to prevent code from being compiled twice.

There are essentially three uses of the preprocessor--directives, constants, and macros. Directives are commands that tell the preprocessor to skip part of a file, include another file, or define a constant or macro. Directives always begin with a sharp sign (#) and for readability should be placed flush to the left of the page. All other uses of the preprocessor involve processing #define'd constants or macros. Typically, constants and macros are written in ALL CAPS to indicate they are special (as we will see).

**Header Files**

The #include directive tells the preprocessor to grab the text of a file and place it directly into the current file. Typically, such statements are placed at the top of a program--hence the name "header file" for files thus included.

**Constants**

If we write

**#define** [identifier name] [value]

whenever [identifier name] shows up in the file, it will be replaced by [value].

If you are defining a constant in terms of a mathematical expression, it is wise to surround the entire value in parentheses:

#define PI\_PLUS\_ONE (3.14 + 1)

By doing so, you avoid the possibility that an order of operations issue will destroy the meaning of your constant:

x = PI\_PLUS\_ONE \* 5;

Without parentheses, the above would be converted to

x = 3.14 + 1 \* 5;

which would result in 1 \* 5 being evaluated before the addition, not after. Oops!

It is also possible to write simply

#define [identifier name]

which defines [identifier name] without giving it a value. This can be useful in conjunction with another set of directives that allow conditional compilation.

**Conditional Compilation**

There are a whole set of options that can be used to determine whether the preprocessor will remove lines of code before handing the file to the compiler. They include #if, #elif, #else, #ifdef, and #ifndef. An #if or #if/#elif/#else block or a #ifdef or #ifndef block must be terminated with a closing #endif.

The #if directive takes a numerical argument that evaluates to true if it's non-zero. If its argument is false, then code until the closing #else, #elif, of #endif will be excluded.

**Macros**

The other major use of the preprocessor is to define macros. The advantage of a macro is that it can be type-neutral (this can also be a disadvantage, of course), and it's inlined directly into the code, so there isn't any function call overhead. (Note that in C++, it's possible to get around both of these issues with templated functions and the inline keyword.)A macro definition is usually of the following form:

#define MACRO\_NAME(arg1, arg2, ...) [code to expand to]

For instance, a simple increment macro might look like this:

#define INCREMENT(x) x++

They look a lot like function calls, but they're not so simple. There are actually a couple of tricky points when it comes to working with macros. First, remember that the exact text of the macro argument is "pasted in" to the macro. For instance, if you wrote something like this:

#define MULT(x, y) x \* y

and then wrote

int z = MULT(3 + 2, 4 + 2);

what value do you expect z to end up with? The obvious answer, 30, is wrong! That's because what happens when the macro MULT expands is that it looks like this:

int z = 3 + 2 \* 4 + 2; // 2 \* 4 will be evaluated first!

So z would end up with the value 13! This is almost certainly not what you want to happen. The way to avoid it is to force the arguments themselves to be evaluated before the rest of the macro body. You can do this by surrounding them by parentheses in the macro definition:

#define MULT(x, y) (x) \* (y)

// now MULT(3 + 2, 4 + 2) will expand to (3 + 2) \* (4 + 2)

It is generally a good idea to surround the macro's code in parentheses if you expect it to return a value. Otherwise, you can get similar problems as when you define a constant. For instance, the following macro, which adds 5 to a given argument, has problems when embedded within a larger statement:

#define ADD\_FIVE(a) (a) + 5

int x = ADD\_FIVE(3) \* 3;

// this expands to (3) + 5 \* 3, so 5 \* 3 is evaluated first

// Now x is 18, not 24!

To fix this, you generally want to surround the whole macro body with parentheses to prevent the surrounding context from affecting the macro body.

#define ADD\_FIVE(a) ((a) + 5)

int x = ADD\_FIVE(3) \* 3;

On the other hand, if you have a multiline macro that you are using for its side effects, rather than to compute a value, you probably want to wrap it within curly braces so you don't have problems when using it following an if statement.

3. HANDLING INPUT AND OUTPUT OPERATIONS

C provides several functions that give different levels of input and output capability. These functions are, in most cases, implemented as routines that call lower-level input/ output functions.

*Classification of Input and Output Operations: non-formatted and formatted functions.*

1. Non Formatted functions-getchar( ),putchar( ),(getch( ),getche( ),putch( ) available only in turbo C)

2. Formatted functions-printf( ), scanf( )

3.1 Nonformatted Input Output Functions

Reading a character

The getchar( ) function:

Single characters can be entered into the computer using the C library function getchar( ). The getchar( ) function is a part of the standard C I/O library. It returns a single character from a standard input device typically a keyboard. The function does not require any arguments, though a pair of empty parentheses must follow the word getchar.

The general syntax

Character variable=getchar();

Where character variable refers to some previously declared character variable.

Example:

A C program contains the following statements

char c;

…………….

c=getchar();

The first statement declares that c is a character-type variable. The second statement causes a single character to be entered from the standard input device and then assigned to c.

If an end-of-file condition is encountered when reading a character with getchar function, the value of the symbolic constant EOF will automatically be returned. (This value will be assigned within the stdio.h file. Typically, EOF will be assigned the value -1, though this may vary from one compiler to another).

The getchar function can also be used to read multi character strings, by reading one character at a time within a multi pass loop.

Writing a character

The putchar( ) function:

Single characters can be displayed using the C library function putchar. This function is complementary to the character input function getchar. The putchar function, like getchar, is a part of the standard C I/O library. It transmits a single character to a standard output device. The character being transmitted will normally be represented as a character type variable. It must be expressed as an argument to the function, enclosed in parentheses, following the word putchar.

The general syntax is

putchar(character variable)

where character variable refers to some previously declared character variable.

A C program contains the following statements

char c;

………

putchar(c);

3.2 Formatted Input and Output Functions

**scanf() function**:

The function scanf() is used to read data into variables from the standard input, namely a keyboard. The general format is:

scanf(format-string, var1,var2,………varn)

Where format-string gives information to the computer on the type of data to be stored in the list of variables var1,var2……varn and in how many columns they will be found.

For example, in the statement:

scanf(“%d %d”, &p, &q);

The two variables in which numbers are used to be stored are p and q. The data to be stored are integers. The integers will be separated by a blank in the data typed on the keyboard.

A sample data line may thus be:

456 18578

Observe that the symbol &(called ampersand) should precede each variable name. Ampersand is used to indicate that the address of the variable name should be found to store a value in it. The manner in which data is read by a scanf statement may be explained by assuming an arrow to be positioned above the first data value. The arrow moves to the next data value after storing the first data value in the storage location corresponding to the first variable name in the list. A blank character should separate the data values.

The scanf statement causes data to be read from one or more lines till numbers are stored in all the specified variable names. The symbol & is very essential in front of the variable name.

If some of the variables in the list of variables in the list of variables in scanf are of type integer and some are float, appropriate descriptions should be used in the format-string.

For example:

scanf(“%d %f %e”, &a , &b, &c);

Specifies that an integer is to be stored in a, float is to be stored in b and a float written using the exponent format in c. The appropriate sample data line is:

485 498.762 6.845e-12

**printf() function**:

The general format of an output function is

printf(format-string, var1,var2…..varn);

Where format-string gives information on how many variables to expect, what type of arguments they are , how many columns are to be reserved for displaying them and any character string to be printed. The printf() function may sometimes display only a message and not any variable value. In the following example:

printf(“Answers are given below”);

The format-string is:

Answers are given below

And there are no variables. This statement displays the format-string on the video display and there are no variables. After displaying, the cursor on the screen will remain at the end of the string. If we want it to move to the next line to display information on the next line, we should have the format-string:

printf(“Answers are given below\n”);

In this string the symbol \n commands that the cursor should advance to the beginning of the next line.

In the following example:

printf(“Answer x= %d \n”, x);

%d specifies how the value of x is to be displayed. It indicates the x is to be displayed as a decimal integer. The variable x is of type int. %d is called the conversion specification and d the conversion character . In the example:

printf(“a= %d, b=%f\n”, a, b);

the variable a is of type int and b of type float or double. % d specifies that a is to be displayed as an integer and %f specifies that, b is to be displayed as a decimal fraction. In this example %d and %f are conversion specifications and d, f are conversion characters.

**Example to indicate how printf() displays answers.**

/\*Program illustrating printf()\*/

# include<stdio.h>

main()

{

int a= 45, b= 67

float x=45.78 , y=34.90

printf(“Output:\n”);

printf(“1,2,3,4,5,6,7,8,0\n”);

printf(“\n”);

printf(“%d, %d,,%f ,%f \n” , a,b,x,y);

printf(“\n”);

}

Output:

1234567890

45,67,45.78,34.90

**Example for illustrating scanf and printf statements**:

/\* Program for illustrating use of scanf and printf statements \*/

#include<stdio.h>

main()

{

int a,b,c,d;

float x,y,z,p;

scanf(“%d %o %x %u”, &a, &b ,&c ,&d);

printf(“the first four data displayed\n”);

printf((“%d %o %x %u \n”, a,b,c,d);

scanf(“%f %e %e %f”, &x, &y, &z, &p);

printf(“Display of the rest of the data read\n”);

printf(“%f %e %e %f\n”, x,y,z,p);

printf(“End of display”);

}

Input:

-768 0362 abf6 3856 -26.68 2.8e-3 1.256e22 6.856

Output:

The first four data displayed

-768 362 abf6 3856

Display of the rest of the data read

-26.680000 2.800000 e-03 1.256000e+22 6.866000

End of display

**Formatted input and output using format specifiers**:

The function scanf is the input analog of printf, providing many of the same conversion facilities in the opposite direction.

int scanf (char \*format, …..)

scanf reads characters from the standard input, interprets them according to the specification in format, and stores the results through the remaining arguments. The format argument is described below; the other arguments, each of which must be a pointer, indicate where the corresponding converted input to be stored.

scanf stops when it exhausts its format string, or when some input fails to match the control specification. It returns as its value the number of successfully matched and assigned input items. This can be used to decide how many items were found . On end of file EOF is returned; note that this is different from 0, which means that the next input character does not match the first specification in the format string. The next call to scanf resumes searching immediately after the last character already converted.

A conversion specification directs the conversion of the next input field. Normally the result is placed in the variable pointed to by the corresponding argument. If assignment suppression is indicated by the \* character, however, the input field is skipped; no assignment is made. An input field is defined as a string of non-white space characters; it extends either to the next white space character or until the field width, if specified, is exhausted. This implies that scanf will read across line boundaries to find its input, since new lines are white space. (White space characters are blank, tab, new line, carriage return, vertical tab and form feed).

**Format Specifiers to read and print data of various types:**

CHARACTER INPUT DATA; ARGUMENT TYPE

d decimal integer; int \*

i integer; int \* The integer may be in octal (leading 0)

or hexadecimal (leading 0x or 0x)

o octal integer(with or without leading zero); int \*.

u unsigned decimal integer; unsigned int \*

x hexadecimal integer (with or without leading ox or ox)

c characters; char \*. The next input characters (default 1) are placed at the indicated spot. The normal skip over

white space is suppressed; to read the next non-white

space character, use of %1s.

s character string (not quoted); char \*, pointing to an

array of characters large enough for the string and a

terminating ‘\0’ that will be added.

e,f,g floating-point number with optional sign, optional

decimal point and optional exponent; float \*.

% literal %; no assignment is made.

The output function printf translates internal values to characters.

The general syntax is

int printf (char \*format, arg1,arg2…………..)

printf converts, formats, and prints its arguments on the standard output under control of the format. It returns the number of characters printed.

The format string contains two types of objects: ordinary characters, which are copied to the output stream, and conversion specifications each of which causes conversion and printing of the next successive argument to printf. Each conversion specification begins with a % and ends with a conversion character. Between the % and the conversion character there may be in order:

* A minus sign, which specifies left adjustment of the converted argument.
* A number that specifies the minimum field width. The converted argument will be printed in a field at least this wide. If necessary it will be padded on the left or right, to make up the field width.
* A period, which separates the field width from the precision.
* A number, the precision, that specifies the maximum number of characters to printed from a string, or the number of digits after the decimal point of a floating point value, or the minimum number of digits for an integer.
* An h if the integer is to be printed as a short, or l if as a long.

CHARACTER ARGUMENT TYPE

d,i int; decimal number

o int; unsigned octal number(without a leading

zero)

x, x int; unsigned hexadecimal number(without a

leading ox or OX), using abcdef or ABCDEF for 10….15

u int; unsigned decimal number

c int; single character.

S char \*; print characters from the string until a ‘\o’ or the number characters given by the

Precision.

F double; [-] m.dddddd, where the number of d’s

Is given by the precision(default 6)

e,E double; [-] m.dddddd e+/-xx or [-] m.dddddd

E+/- xx, where the number of d’s is given

By the precision(default 6)

g,G double; use %e or %E if the exponent is less

than or equal to the precision; otherwise use

%f. Trailing zeros and a trailing decimal

point are not printed.

P void \*; pointer(implementation-dependent

Representation).

% no argument is converted ; print a %

A width or precision may be specified as \* , in which case the value is computed by converting the next argument(which must be an int). For example, to print at most max characters from a string s.

Printf (“%.\*s”, max, s);

Most of the format conversions have been illustrated in earlier . One exception is precision as it relates to strings. The following example shows the effect of a variety of specifications in printing “hello world”.

:%s: :hello, world:

:%10s: :hello, world:

:%.10s: :hello, wor:

:%-10s: :hello, world:

:%.15s: :hello, world:

:%-15s: :hello, world:

:%15.10s: : hello, wor:

:%-15.10s: :hello, wor :

**Address Operator**:

The unary & operator takes the address of its operand. The operand must be an lvalue referring neither to a bit-field not to an object declared as register, or must be of function type. The result is a pointer to the object or function referred to by the lvalue. If the type of the operand is T, the type of the result is “pointer to T”.

**Programming Examples:**

**Example 1**

The program in 1 shows the use of **getchar** function in an interactive environment.

The program displays a question of YES/NO type to the user and reads the user's response in a single character (Y or N). If the response is Y, it outputs the message

My name is BUSY BEE

otherwise, outputs.

You are good for nothing

Note there is one line space between the input text and output message.

READING A **CHARACTER** FROM KEYBOARD

**Program**

#include <stdio.h>

main()

{

char answer;

printf("Would you like to know my name?\n");

printf("Type Y for YES and N for NO: ");

answer = getchar(); /\* .... Reading a character...\*/

if(answer == 'Y' || answer == 'y')

printf("\n\nMy name is BUSY BEE\n");

else

printf("\n\nYou are good for nothing\n");

}

***Output***

Would you like to know my name?

Type Y for YES and N for NO: Y

My name is BUSY BEE

Would you like to know my name?

Type Y for YES and N for NO: n

You are good for nothing

Example 1 **Use of** getchar **function**

**Example 2**

The program of Example 2 requests the user to enter a character and displays a message on the screen telling the user whether the character is an alphabet or digit, or any other special character.

This program receives a character from the keyboard and tests whether it is a letter or digit and prints out a message accordingly. These tests are done with the help of the following functions:

**isalpha(character)**

**isdigit(character)**

For example, **isalpha** assumes a value non-zero (TRUE) if the argument **character** contains an alphabet; otherwise it assumes 0 (FALSE). Similar is the case with the function **isdigit**.

TESTING CHARACTER TYPE

**Program:**

#include <stdio.h>

#include <ctype.h>

main()

{

char character;

printf("Press any key\n");

character = getchar();

if (isalpha(character) > 0)

printf("The character is a letter.");

else

if (isdigit (character) > 0)

printf("The character is a digit.");

else

printf("The character is not alphanumeric.");

}

***Output***

Press any key

h

The character is a letter.

Press any key

5

The character is a digit.

Press any key

\*

The character is not alphanumeric.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Example 2 **Program to test the** characte**r type**

**Example 3**

A program that reads a character from keyboard and then prints it in reverse case is given in Example 3. That is, if the input is upper case, the output will be lower case and vice versa.

The program uses three new functions: **islower, toupper**, and **tolower**. The function **islower** is a conditional function and takes the value TRUE if the argument is a lower case alphabet; otherwise takes the value FALSE. The function **toupper** converts the lower case argument into an upper case alphabet while the function **tolower** does the reverse.

WRITING A **CHARACTER** TO THE SCREEN

*Program*

#include <stdio.h>

#include <ctype.h>

main()

{

char alphabet;

printf("Enter an alphabet");

putchar('\n'); /\* move to next line \*/

alphabet = getchar();

if (islower(alphabet))

putchar(toupper(alphabet));

else

putchar(tolower(alphabet));

}

***Output***

Enter an alphabet

a

A

Enter an alphabet

Q

q

Enter an alphabet

z

Z

***Example 3*** *Reading and writing of alphabets in reverse case*

**Example 4**

Various input formatting options for reading integers are experimented in the program shown in Example 4.

The first **scanf** requests input data for three integer values **a, b,** and **c,** and accordingly three values 1, 2, and 3 are keyed in. Because of the specification %\*d the value 2 has been skipped and 3 is assigned to the variable **b**. Notice that since no data is available for c, it contains garbage.

The second **scanf** specifies the format %2d and %4d for the variables **x** and **y** respectively. Whenever we specify field width for reading integer numbers, the input numbers should not contain more digits that the specified size. Otherwise, the extra digits on the right-hand side will be truncated and assigned to the next variable in the list. Thus, the second **scanf** has truncated the four digit number 6789 and assigned 67 to **x** and 89 to **y**. The value 4321 has been assigned to the first variable in the immediately following **scanf** statement.

READING **INTEGER** NUMBERS

**Program:**

main()

{

int a,b,c,x,y,z;

int p,q,r;

printf("Enter three integer numbers\n");

scanf("%d %\*d %d",&a,&b,&c);

printf("%d %d %d \n\n",a,b,c);

printf("Enter two 4-digit numbers\n");

scanf("%2d %4d",&x,&y);

printf("%d %d\n\n", x,y);

printf("Enter two integers\n");

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

printf("%d %d \n\n",a,x);

printf("Enter a nine digit number\n");

scanf("%3d %4d %3d",&p,&q,&r);

printf("%d %d %d \n\n",p,q,r);

printf("Enter two three digit numbers\n");

scanf("%d %d",&x,&y);

printf("%d %d",x,y);

}

*Output*

Enter three integer numbers

1 2 3

1 3 -3577

Enter two 4-digit numbers

6789 4321

67 89

Enter two integers

44 66

4321 44

Enter a nine-digit number

123456789

66 1234 567

Enter two three-digit numbers

123 456

89 123

***Example 4*** *Reading integers using* **scanf**

**Example 5**

Reading of real numbers (in both decimal point and exponential notation) is illustrated in Example 5.

READING OF **REAL** NUMBERS

**Program:**

main()

{

float x,y;

double p,q;

printf("Values of x and y:");

scanf("%f %e", &x, &y);

printf("\n");

printf("x = %f\ny = %f\n\n", x, y);

printf("Values of p and q:");

scanf("%lf %lf", &p, &q);

printf("\np = %lf\nq = %e",p,q);

printf("\n\np = %.12lf\np = %.12e", p,q);

}

*Output*

Values of x and y:12.3456 17.5e-2

x = 12.345600

y = 0.175000

Values of p and q:4.142857142857 18.5678901234567890

p = 4.142857142857

q = 1.856789012346e+001

Example 5 **Reading of** real **numbers**

**Example 6**

Reading of strings using **%wc** and **%ws** is illustrated in Example 6.

The program in Example 6 illustrates the use of various field specifications for reading strings. When we use **%wc** for reading a string, the system will wait until the *w*th character is keyed in.

Note that the specification **%s** terminates reading at the encounter of a blank space. Therefore, **name2** has read only the first part of "New York" and the second part is automatically assigned to **name3**. However, during the second run, the string "New-York" is correctly assigned to **name2**.

READING STRINGS

**Program**

main()

{

int no;

char name1[15], name2[15], name3[15];

printf("Enter serial number and name one\n");

scanf("%d %15c", &no, name1);

printf("%d %15s\n\n", no, name1);

printf("Enter serial number and name two\n");

scanf("%d %s", &no, name2);

printf("%d %15s\n\n", no, name2);

printf("Enter serial number and name three\n");

scanf("%d %15s", &no, name3);

printf("%d %15s\n\n", no, name3);

}

*Output*

Enter serial number and name one

1 123456789012345

1 123456789012345r

Enter serial number and name two

2 New York

2 New

Enter serial number and name three

2 York

Enter serial number and name one

1 123456789012

1 123456789012 r

Enter serial number and name two

2 New-York

2 New-York

Enter serial number and name three

3 London

3 London

Example 6 **Reading of strings**

**Example 7**

The program in Example 7 illustrates the function of %[ ] specification.

ILLUSTRATION OF %[ ] SPECIFICATION

***Program A***

main()

{

char address[80];

printf("Enter address\n");

scanf("%[a-z]", address);

printf("%-80s\n\n", address);

}

*Output*

Enter address

new delhi 110002

new delhi

ILLUSTRATION OF %[^ ] SPECIFICATION

***Program-B***

main()

{

char address[80];

printf("Enter address\n");

**scanf("%[^\n]", address);**

**printf("%-80s", address);**

**}**

*Output*

Enter address

New Delhi 110 002

New Delhi 110 002

***Example 7*** *Illustration of conversion specification%[] for strings*

**Example 8**

The program presented in Example 8 illustrates the testing for correctness of reading of data by**scanf** function.

The function **scanf** is expected to read three items of data and therefore, when the values for all the three variables are read correctly, the program prints out their values. During the third run, the second item does not match with the type of variable and therefore the reading is terminated and the error message is printed. Same is the case with the fourth run.

In the last run, although data items do not match the variables, no error message has been printed. When we attempt to read a real number for an **int** variable, the integer part is assigned to the variable, and the truncated decimal part is assigned to the next variable.

Note that the character `2' is assigned to the character variable c.

TESTING FOR CORRECTNESS OF INPUT DATA

**Program**

main()

{

int a;

float b;

char c;

printf("Enter values of a, b and c\n");

if (scanf("%d %f %c", &a, &b, &c) == 3)

printf("a = %d b = %f c = %c\n" , a, b, c);

else

printf("Error in input.\n");

}

**Output** Enter values of a, b and c

12 3.45 A

a = 12 b = 3.450000 c = A

Enter values of a, b and c

23 78 9

a = 23 b = 78.000000 c = 9

Enter values of a, b and c

8 A 5.25

Error in input.

Enter values of a, b and c

Y 12 67

Error in input.

Enter values of a, b and c

15.75 23 X

***a = 15 b = 0.750000 c = 2***

***Example 8*** *Detection of errors in* ***scanf*** *input*

**Example 9**

The program in Example 9 illustrates the output of integer numbers under various formats.

PRINTING OF **INTEGER** NUMBERS

**Program:**

main()

{

int m = 12345;

long n = 987654;

printf("%d\n",m);

printf("%10d\n",m);

printf("%010d\n",m);

printf("%-10d\n",m);

printf("%10ld\n",n);

printf("%10ld\n",-n);

}

*Output*

12345

12345

0000012345

12345

987654

-987654

***Example 9*** *Formatted output of integers*

**Example 10**

All the options of printing a real number are illustrated in Example 10.

PRINTING OF **REAL NUMBERS**

***Program***:

main()

{

float y = 98.7654;

printf("%7.4f\n", y);

printf("%f\n", y);

printf("%7.2f\n", y);

printf("%-7.2f\n", y);

printf("%07.2f\n", y);

printf("%\*.\*f", 7, 2, y);

printf("\n");

printf("%10.2e\n", y);

printf("%12.4e\n", -y);

printf("%-10.2e\n", y);

printf("%e\n", y);

}

**Output** 98.7654

98.765404

98.77

98.77

0098.77

98.77

9.88e+001

-9.8765e+001

9.88e+001

***9.876540e+001***

***Example 10*** *Formatted output of* ***real*** *numbers*

**Example 11**

Printing of characters and strings is illustrated in Example11.

PRINTING OF CHARACTERS AND STRINGS

**Program**

main()

{

char x = 'A';

static char name[20] = "ANIL KUMAR GUPTA";

printf("OUTPUT OF CHARACTERS\n\n");

printf("%c\n%3c\n%5c\n", x,x,x);

printf("%3c\n%c\n", x,x);

printf("\n");

printf("OUTPUT OF STRINGS\n\n");

printf("%s\n", name);

printf("%20s\n", name);

printf("%20.10s\n", name);

printf("%.5s\n", name);

printf("%-20.10s\n", name);

printf("%5s\n", name);

}

***Output***

OUTPUT OF CHARACTERS

A

A

A

A

A

OUTPUT OF STRINGS

ANIL KUMAR GUPTA

ANIL KUMAR GUPTA

ANIL KUMAR

ANIL

ANIL KUMAR

ANIL KUMAR GUPTA

***Example 11*** *Printing of characters and strings*

**Sample C programs**:

1. **Program to demonstrate printf statement**

#include<stdio.h>

main()

{

printf(“hello, world\y”);

printf(“hello, world\7”);

printf(“hello, world\?”);

}

1. **Program to convert farenheit to Celsius**

#include<stdio.h>

main()

{

float fahr, Celsius;

printf(“ enter the value for farenheit\n”);

scanf(“ %f”, &fahr);

Celsius=(5.0/9.0)\*fahr-32.0;

printf(“%f %f \n”, fahr,Celsius);

}

1. **Program to depict interactive computing using scanf function**.

#include<stdio.h>

main()

{

int number;

printf(“enter an integer number\n”);

scanf(“%d”, &number);

If (number<100)

{

printf(“Your number is smaller than 100\n\n”);

else

printf(“Your number contains more than two digits\n”);

}

Output

Enter an integer number

54

Your number is smaller than 100

Enter an integer number

108

Your number contains more than digits

1. **Program to depict interactive investment program**

#include<stdio.h>

main()

{

int year,period;

float amount,inrate,value;

printf(“Input amount , interest rate and period \n\n”);

scanf(“%f %f %d”, &amount, &inrate,&period);

printf(“\n”);

year=1;

while(year<=period)

{

value amount + inrate\*amount;

printf(“%2d Rs. %8.2f\n”, year, value);

amount=value;

year=year+1;

}

}

1. **Program to calculate the average of a set of N numbers**

#define N 10

main()

{

int count;

float sum, average,number;

sum=0;

count=0;

while(count<N)

{

scanf(“%f”, &number);

sum=sum+number;

count=count+1;

}

average= sum/N;

printf(“N=%d Sum= %f”, N, sum);

printf(“Average=%f”, average);

}

6) **Program to convert days to months and days**

#include<stdio.h>

main()

{

int months,days;

printf(“enter days \n”);

scanf(“%d”, &days);

months=days/30;

days=days%30;

printf(“Months = %d Days= %d”, months,days);

}

4. OPEARTORS

An operator is a symbol that specifies the mathematical, logical or relational operation to be performed. An expression is any computation that yields a value.

C supports a rich set of operators. Operators are used in programs to manipulate data and variables. Operators can be classified into a number of categories. They are:

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Assignment operators
5. Increment and Decrement operators
6. Conditional operators
7. Bit wise operators
8. Special operators

4.1 Arithmetic operators

C provides all the basic arithmetic operators. The operators +,-,\* and / all work the same way as they do in other languages. These can operate on any built-in data type allowed in C. The unary minus operator, in effect, multiplies its single operand by -1. Therefore, a number preceded by a minus sign changes its sign.

C provides all the basic arithmetic operators. They are:

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| + | Addition or Unary plus |
| - | Subtraction or Unary minus |
| \* | Multiplication |
| / | Division |
| % | Modulo division |

Integer division truncates any fractional part. The Modulo division operator produces the remainder of an integer division. Examples or use of arithmetic operators are:

a-b a+b

a\*b a/b

a%b -a\*b

Here a and b are variables and are known as operands. The modulo divison operator % cannot be used on floating point data.

**4.2 Relational operators**

We often compare two quantities and depending on their relation, to take certain decisions. For example, we may compare the age of two persons, or the price of two items, and so on. These comparisons can be done with the help of relational operators.

The value of relational expression is either one or zero. It is one if the specified relation is true and zero if the relation is false.

For example,

10 < 20 is true

20 < 10 is false

C supports six relational operators in all. These operators and their meanings are shown below

Relational Operators

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| < | Less Than |
| <= | Less than or equal to |
| > | Greater than |
| >= | Greater than or equal to |
| == | Equal to |
| != | Not equal to |

Given below are some examples of simple relational expressions and their values:

4.5<= 10 TRUE

4.5<10 FALSE

-35>= 0 FALSE

10<7+5 TRUE

a+b == c+d TRUE only if the sum of values of a and b is equal to the

sum of values of c and d.

When arithmetic expressions are used on either side of a relational operator, the arithmetic expressions will be evaluated first and then the results compared. That is, arithmetic operators have a higher priority over relational operators.

Relational expressions are used in decision statements such as, if and while to decide the course of action of a running program.

4.3 Logical operators

&&(AND),||(OR),^(EX-OR) and !(NOT) are different logical operators.

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| && | Logical AND |
| || | Logical OR |
| ! | Logical NOT |
| ^ | Logical XOR |

The logical operators && and || are used when we want to test more than one condition and make decisions.

For example,

A > b && x == 10

This is true only if a > b is true and x == 10 is true. If either of them false, the expression is false.

**Truth table for AND (&&)**

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **A && B** |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**Truth table for OR (||)**

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **A || B** |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**Truth table for NOT (!)**

|  |  |
| --- | --- |
| **A** | **!A** |
| 0 | 1 |
| 1 | 0 |

**Truth table for XOR (^)**

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **A ^ B** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

4.4 Assignment operators

Assignment operators are used to assign the result of an expression to a variable. The form is

v =exp;

where v is a variable, exp is an expression. For example,

x=10; x=x+3; x=x+a\*b;

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| = | Assignment |

**Short hand assignment operators:**

In addition to the usual assignment operator ‘=’ C has a set of assignment operators called shorthand operators.

Some of the shorthand operators are listed below:

=

+=

-=

\*=

/=

%=

>>=

<<=

&=

^=

!=

for example,

x=x+3

this statement is equivalent to

x+=3;

when this statement is executed, 3 is added to x.

4.5 Increment(++) and Decrement(--) operators

Increment operator ++

The operator ++ adds 1 to the operand. Two types of increment operators are:

Prefix Increment operator(++op)

This first adds 1 to the operand and then the result is assigned to the variable on left.

For example,

m=5;

++m;

This statement increases the value of m by 1. that is m is 6.

Postfix Increment operator (op++)

Postfix operator first assigns the value to the variable on left and then increments the operand.

For example,

m=5;

m++;

This statement increases the value of m by 1. That is m is 6.

While ++m and m++ mean the same thing when they form statements independently. They behave differently when they are used in expressions on the right hand side of assignment statements.

For example,

m=5;

Y=++m;

In this case, the value of y and m would be 6. Suppose, if we rewrite the above statements as

m=5;

y=m++;

here, the value of y would be 5 and m would be 6.

4.6 Conditional operator (? :)

This is also called as a ternary operator. The general form is:

exp1 ? exp2 : exp3;

where exp1, exp2 and exp3 are expressions.

Here, exp1 is evaluated first. If it is nonzero (true), then the expression exp2 is evaluated and becomes the values of the expression. If exp1 is false, exp3 is evaluated and its value becomes the values of the expression. Note that only one of the expressions either exp2 or exp3 is evaluated. Forexample,  
 a=10;

b=34;

max=(a > b) ? a: b;

In this example, max will be assigned the value of b because the condition is false. So, exp3 (b) is evaluated and its value is assigned to max.

4.7 Bitwise operators

These operators are used for testing the bits, or shifting them left or right. These Operators may not be applied to float or double.

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| & | bitwise AND |
| | | bitwise OR |
| ^ | bitwise XOR |
| << | shift left |
| >> | shift right |

**Bitwise AND**

The result of bitwise AND operator is 1 if both the bits have a value of 1; otherwise it is 0. For example,

if x=13 and y=25 then

z= x & y;

if we execute this statement,

x=0000 0000 0000 1101

y=0000 0000 0001 1001

the result would be:

z=0000 0000 000 1001

**Bitwise OR**

The result of bitwise OR operator is 1 if atleast one of the bits has a value of 1; otherwise it is 0. For example,

if x=13 and y=25 then

z= x | y;

if we execute this statement,

x=0000 0000 0000 1101

y=0000 0000 0001 1001

the result would be:

z=0000 0000 001 1101

**Bitwise XOR (Exclusive OR)**

The result of bitwise XOR operator is 1 if only one of the bits is 1; otherwise it is 0. For example,

if x=13 and y=25 then

z= x ^y;

if we execute this statement,

x=0000 0000 0000 1101

y=0000 0000 0001 1001

the result would be:

z=0000 0000 000 0100

**Bitwise shift operators**

The shift operators are used to move bit patterns either to the left or to the right. The shift operators are represented by the symbols << and >>.

**Left shift**

**o**p << n

op is an integer expression that is to be shifted and n is the number of bit positions to be shifted. The left shift operation shifts all the bits in the operand op to be shifted to the left by n positions. The leftmost n bits in the original bit pattern will be lost and rightmost n bits that are vacated will be filled with 0s.

For example,

x is an unsigned integer that has the value 11. Then bit pattern of x is:

0000 1011

x << 1 is,

0001 0110 that is 22.

This is equal to the value of x will be multiplied by 2.

**Right shift**

op >> n

The right shift operation shifts all the bits in the operand op to be shifted to the right by n positions. The rightmost n bits in the original bit pattern will be lost and leftmost n bits that are vacated will be filled with 0s if the op is an unsigned integer. If the variable to be shifted is signed, then the operation is machine dependent.

For example,

x is an unsigned integer that has the value 11. Then bit pattern of x is:

0000 1011

x >> 1 is,

0000 0101 that is 5.

This is equal to the value of x will be divided by 2.

**Bitwise complement operator (~)**

The operator ‘~’ inverts all the bits represented by its operand. That is, 0s become 1s and 1s become 0.

For example,

x=10;

binary representation of x is=0000 1010

~x =1111 0101

4.8 Special Operators

**sizeof operator**

The sizeof is a compile time operator and when used with an operand, it returns the number of bytes the operand occupies. The operand may be a variable, a constant or a data type qualifier.

For example,

int n;

m=sizeof(n);

Will return 2. That is m is assigned a value 2, because int occupies 2 bytes.

m=sizeof(34.76);

Will return 4. That is m is assigned a value 4, because int occupies 4 bytes.

**Comma Operator**

C has some special operators. The comma operator is one among them. This operator can be used to link the related expressions together. A comma-linked list of expressions are evaluated left to right and the value of right-most expression is the value of the combined expression.

For example, the statement

Value=(a=2, b=6 ,a+b);

First assigns the value 2 to a, then assigns 6 to b, and finally assigns 8(i.e 2+6) to value. The comma operator has the lowest precedence of all operators,hence the parentheses are necessary.

Some examples are given below:

In for loops:

For(a=1, b=10;a<=b; a++, b++)

In while loops:

While(c=getchar(), C!=’10’)

Exchanging values:

T=x, x=y, y=t;