**INTRODUCTION TO COMPUTER SCIENCE**

**Course code: GST 131**

KPOROSTIC BOY ft TERRO FANCY

**Class tests and Assignments**

Class tests and Assignments: At least one Class tests and one quiz shall contribute 35% to the final mark.

Attendance at all lectures and tests is compulsory. This shall contribute 5% to the total final mark.

**Semester Examinations:** The end of Semester Examination shall contribute 60% to the final mark.

**Short course description:**

This course is a foundation course for computer science, Mathematics and physics students. The course traces the history of computer, explain the functional component of computer, introduces application of computer, programming process as well as problem solving techniques.

**Learning outcomes:**

By the end of the course, students should be able

* To be able to define *computer program*, *algorithm*, and *high-level programming language*.
* To be able to list the basic stages involved in writing a computer program.
* To be able to list the major components of a computer and describe how they work together.
* To be able to distinguish between hardware and software.
* To be able to apply an appropriate problem-solving method for developing an algorithmic solution to a

problem.

**Teaching methods:**

Class lectures, group discussions, take-home assignments

**Syllabus**

History of computers, functional components of computer, characteristics of a computer, problem solving, flow charts, Algorithms, computer programming, Statements, symbolic names, Arrays, subscripts, expressions and control statements. Introduction to BASIC OR FORTRAN programming language, computer applications.

**Reading list:**

**Prescribed textbook:**

**Computer studies for Beginners by**

**Adekunle O Eyitayo, Oduronke T. Eyitayo and Olufunmilayo M. Akejj**

**ISBN 9-78-330296-5**

**Definition of a Computer**

A computer is an electronic device, use for manipulating data according to a list of instructions. A computer is a machine that is used to process data. In a more formal definition, a computer is any machine or device which, under the control of a stored program, can accept data in a prescribed form, process the data, and supply the results as information in a specified form.

**History of Computers**

The natural computing device man naturally has is the human brain. Man has an inbuilt capability to be able to carry out limited calculation, manipulation and express certain level of interpretations. The limitation of man in this capacity make him to think of devices that can be of assistance. Foremost of this device are Fingers and Toes, small stones and pegs.

The first early aids of counting were the fingers and toes. Man uses his fingers to help him count, it was a very simple device which help man to carry out simple additions. The fact that the total number that can be counted with fingers and toes is twenty make man to device another means like stones or small sticks

In the early days, when man wants to count say the number of animals he owned, he built a pile of stones with one stone for every animal. After he had finished building the piles he needed, he then needed to look at it and see how many there were on the piles he gathered. This method was used to count larger numbers. The stones could be used to count hundreds of numbers.

Abacus is the next idea which was made of coloured beads threaded on a string. At first the beads were made of stones. Around the year 1200, the Chinese used this method to count. The beads were threaded on lines of wire frame. The beads on the first line counted the units. The beads on the second wire counted the hundreds and so on. By moving the beads back and forth along wires, higher numbers could be added. This device was called an ABACUS. An Abacus is a wooden frame with strings on which colourful beads are placed; a picture of this is shown below.



After abacus, a lot of technical improvement and advancement took place before arriving at what can be called a computer. From Abacus the development of computer moves through Pascaline machine, stepped reckoner, Jacquard’s loam, babbages’s engine, Hermann Hollerith’s Machine and finally through Generation of Computers .

Each generation of computer is characterized by a major technological development that fundamentally changed the way computers operate, resulting in increasingly smaller, cheaper, more powerful, more efficient and reliable devices. The table below reflects the generation of computer and their major components.

|  |  |  |
| --- | --- | --- |
| Generation of Computer | Major Components | Characteristics |
| First Generation | Vacuum tubes | Very large, very noisy |
| Second Generation | Transistors | large, fairly noisy |
| Third Generation | Integrated circuits | Smaller and interactive system with introduction of keyboards, monitors and operating system |
| Fourth Generation | Microprocessors | Graphical User interface, networking features, mouse, hand held devices |
| Fifth generation | Multi processors, superconductors | Parallel progr. Artificial and intelligence computing, voice recognition |
|  |  |  |

**Characteristics of computer**

The characteristics of computers that have made them so powerful and universally useful are speed, accuracy, diligence, versatility and storage capacity.

Speed: Computers work at an incredible speed. A powerful computer is capable of performing about 3-4 million simple instructions per second.

Accuracy: In addition to being fast, computers are also accurate. Errors that may occur can almost always be attributed to human error (inaccurate data, poorly designed system or faulty instructions/programs written by the programmer)

Diligence: Unlike human beings, computers are highly consistent. They do not suffer from human traits of boredom and tiredness resulting in lack of concentration. Computers, therefore, are better than human beings in performing voluminous and repetitive jobs.

Versatility: Computers are versatile machines and are capable of performing any task as long as it can be broken down into a series of logical steps. The presence of computers can be seen in almost every sphere – Railway/Air reservation, Banks, Hotels, Weather forecasting and many more.

Storage Capacity: Today’s computers can store large volumes of data. A piece of information once recorded (or stored) in the computer, can be retrieved almost instantaneously.

**COMPUTER SCIENCE**: is the science of computational technology and it covers three broad areas. Hardware, software and humanware / peoplewear.

Hardware/Functional Components

A computer system consists of mainly four basic units (See the fig. below); namely input unit, storage unit, central processing unit and output unit. Central Processing unit further includes Arithmetic & logic unit and control unit.

Control unit

ALU

Memory

A computer performs five major operations or functions irrespective of its size and make. These are • it accepts data or instructions as input, • it stores data and instruction • it processes data as per the instructions, • it controls all operations inside a computer, and • it gives results in the form of output.

When a program is executing, the computer proceeds through a series of steps, the *fetch-execute cycle*:

1. The control unit retrieves (*fetches*) the next coded instruction from memory.

2. The instruction is translated into control signals.

3. The control signals tell the appropriate unit (arithmetic/logic unit, memory, I/O device) to perform (*execute*) the instruction. 4. The sequence repeats from Step 1.

a). Processing Unit: The task of performing operations like arithmetic and logical operations is called processing. The Central Processing Unit (CPU) takes data and instructions from the storage unit and makes all sorts of calculations based on the instructions given and the type of data provided. It is then sent back to the storage unit. CPU includes Arithmetic logic unit (ALU) and control unit (CU)

• Arithmetic Logic Unit: All calculations and comparisons, based on the instructions provided, are carried out within the ALU. It performs arithmetic functions like addition, subtraction, multiplication, division and also logical operations like greater than, less than and equal to etc.

• Control Unit: Controlling of all operations like input, processing and output are performed by control unit. It orders the step by step processing of all operations inside the computer.

b) Memory Unit: this is the medium of storage of a computer system. Computer’s memory can be classified into two types; primary memory and secondary memory

i). Primary Memory can be further classified as RAM and ROM.

• RAM or Random Access Memory is a unit of memory in a computer system. It is the place in a computer where the operating system, application programs and the data in current use are kept temporarily so that they can be accessed by the computer’s processor. It is said to be ‘volatile’ since its contents are accessible only as long as the computer is on. The contents of RAM are no longer available once the computer is turned off.

ROM or Read Only Memory is a special type of memory which can only be read and contents of which are not lost even when the computer is switched off. It typically contains manufacturer’s instructions. Among other things, ROM also stores an initial program called the ‘bootstrap loader’ whose function is to start the operation of computer system once the power is turned on.

ii. Secondary Memory

RAM is volatile memory having a limited storage capacity. Secondary/auxiliary memory is storage other than the RAM. These include devices that are peripheral and are connected and controlled by the computer to enable permanent storage of programs and data.

Secondary storage devices are of two types; magnetic and optical. Magnetic devices include hard disks and optical storage devices are CDs, DVDs, Pen drive, Zip drive etc.

• Hard Disk

Hard disks are made up of rigid material and are usually a stack of metal disks sealed in a box. The hard disk and the hard disk drive exist together as a unit and is a permanent part of the computer where data and programs are saved. These disks have storage capacities ranging from 1GB to 80 GB and more. Hard disks are rewritable.

• Compact Disk

Compact Disk (CD) is portable disk having data storage capacity between 650-700 MB. It can hold large amount of information such as music, full-motion videos, and text etc. CDs can be either read only or read/write type.

• Digital Video Disk

Digital Video Disk (DVD) is similar to a CD but has larger storage capacity and enormous clarity. Depending upon the disk type it can store several Gigabytes of data. DVDs are primarily used to store music or movies and can be played back on your television or the computer too.

c) Input/output Devices: These devices are used to enter information and instructions into a computer for storage or processing and to deliver the processed data to a user. Input/output devices are required for users to communicate with the computer. In simple terms, input devices bring information INTO the computer and output devices bring information OUT of a computer system. These input/output devices are also known as peripherals since they surround the CPU and memory of a computer system.

Input Devices an input device is any device that provides input to a computer. There are many input devices, but the two most common ones are a keyboard and mouse. Every key you press on the keyboard and every movement or click you make with the mouse sends a specific input signal to the computer.

• Keyboard: The keyboard is very much like a standard typewriter keyboard with a few additional keys. The basic layout of characters is maintained to make it easy to use the system. The additional keys are included to perform certain special functions. These are known as function keys that vary in number from keyboard to keyboard.

• Mouse: A device that controls the movement of the cursor or pointer on a display screen. A mouse is a small object you can roll along a hard and flat surface. Its name is derived from its shape, which looks a bit like a mouse. As you move the mouse, the pointer on the display screen moves in the same direction.

• Trackball: A trackball is an input device used to enter motion data into computers or other electronic devices. It serves the same purpose as a mouse, but is designed with a moveable ball on the top, which can be rolled in any direction.

• Touchpad: A touch pad is a device for pointing (controlling input positioning) on a computer display screen. It is an alternative to the mouse. Originally incorporated in laptop computers, touch pads are also being made for use with desktop computers. A touch pad works by sensing the user’s finger movement and downward pressure.

• Touch Screen: It allows the user to operate/make selections by simply touching the display screen. A display screen that is sensitive to the touch of a finger or stylus, monitors and industrial control panels.

• Light Pen: Light pen is an input device that utilizes a light-sensitive detector to select objects on a display screen.

• Magnetic ink character recognition (MICR): MICR can identify character printed with a special ink that contains particles of magnetic material. This device particularly finds applications in banking industry.

• Optical mark recognition (OMR): Optical mark recognition, also called mark sense reader is a technology where an OMR device senses the presence or absence of a mark, such as pencil mark. OMR is widely used in marking tests such as aptitude test.

• Scanner: Scanner is an input device that can read text or illustration printed on paper and translates the information into a form that the computer can use. A scanner works by digitizing an image.

Output Devices:

Output device receives information from the CPU and presents it to the user in the desired form. The processed data, stored in the memory of the computer is sent to the output unit, which then converts it into a form that can be understood by the user. The output is usually produced in one of the two ways – on the display device, or on paper (hard copy).

• Monitor: is often used synonymously with “computer screen” or “display.” Monitor is an output device that resembles the television screen. The monitor is associated with a keyboard for manual input of characters and displays the information as it is keyed in. It also displays the program or application output. Like the television, monitors are also available in different sizes.

• Printer: Printers are used to produce paper (commonly known as hardcopy) output. Based on the technology used, they can be classified as Impact or Non-impact printers.

Impact printers use the typewriting printing mechanism wherein a hammer strikes the paper through a ribbon in order to produce output. Dot-matrix and Character printers fall under this category.

Non-impact printers do not touch the paper while printing. They use chemical, heat or electrical signals to etch the symbols on paper. Inkjet, Deskjet, Laser, Thermal printers fall under this category of printers.

• Plotter: Plotters are used to print graphical output on paper. It interprets computer commands and makes line drawings on paper using multicoloured automated pens. It is capable of producing graphs, drawings, charts, maps etc.

• Facsimile (FAX): Facsimile machine, a device that can send or receive pictures and text over a telephone line. Fax machines work by digitizing an image.

• Sound cards and Speaker(s): An expansion board that enables a computer to manipulate and output sounds. Sound cards are necessary for nearly all CD-ROMs and have become commonplace on modern personal computers. Sound cards enable the computer to output sound through speakers connected to the board, to record sound input from a microphone connected to the computer, and manipulate sound stored on a disk.

SOFTWARE

Computer software is the set of programs that makes the hardware perform a set of tasks in particular order. Hardware and software are complimentary to each other. Both have to work together to produce meaningful results. Computer software is classified into two broad categories; system software and application software.

* Applications

Software that can perform a specific task for the user, such as word processing, accounting, budgeting or payroll, fall under the category of application software. Word processors, spreadsheets, database management systems are all examples of general purpose application software.

Types of application software are:

• Word processing software: The main purpose of this software is to produce documents. MS-Word, Word Pad, Notepad and some other text editors are some of the examples of word processing software.

• Database software: Database is a collection of related data. The purpose of this software is to organize and manage data. The advantage of this software is that you can change way data is stored and displayed. MS access, dBase, FoxPro, Paradox, and Oracle are some of the examples of database software.

• Spread sheet software: The spread sheet software is used to maintain budget, financial statements, grade sheets, and sales records. The purpose of this software is organizing numbers. It also allows the users to perform simple or complex calculations on the numbers entered in rows and columns. MS-Excel is one of the examples of spreadsheet software.

• Presentation software: This software is used to display the information in the form of slide show. The three main functions of presentation software is editing that allows insertion and formatting of text, including graphics in the text and executing the slide shows. The best example for this type of application software is Microsoft PowerPoint.

• Multimedia software: Media players and real players are the examples of multimedia software. This software will allow the user to create audio and videos. The different forms of multimedia software are audio converters, players, burners, video encoders and decoders

* System software

System software consists of a group of programs that control the operations of a computer equipment including functions like managing memory, managing peripherals, loading, storing, and is an interface between the application programs and the computer. MS DOS (Microsoft’s Disk Operating System), UNIX are examples of system software.

c) Programmable Software- Computer language or programming language is a coded syntax used by computer programmers to communicate with a computer. Computer language establishes a flow of communication between software programs. The language enables a computer user to dictate what commands the computer must perform to process data. These languages can be classified into following categories.

1. Machine language

2. Assembly language

3. High level language

High Level Languages are user-friendly languages which are similar to English with vocabulary of words and symbols. These are easier to learn and require less time to write. They are problem oriented rather than ‘machine’ based. A compiler is a special program that processes statements written in a particular programming language called source code and converts them into machine language or “machine code” that a computer’s processor uses. Compiler translates high level language programs directly into machine language program. This process is called compilation.

Computer Programming

Computer programming (often shortened to programming, scripting, or coding) is the process of designing, writing, testing, debugging, and maintaining the source code of computer programs. This source code is written in one or more programming languages (such as C++, C#, Java etc.). The purpose of programming is to create a set of instructions that computers use to perform specific operations or to exhibit desired behaviors. The process of writing source code often requires expertise in many different subjects, including knowledge of the application domain, specialized algorithms and formal logic.

How Do We Write a Program? (Programming process)

A computer is not intelligent. It cannot analyze a problem and come up with a solution. A human (the *programmer*) must analyze the problem, develop the instructions for solving the problem, and then have the computer carry out the instructions. What’s the advantage of using a computer if it cannot solve problems? Once we have written a solution for the computer, the computer can repeat the solution very quickly and consistently, again and again. The computer frees people from repetitive and boring tasks.

To write a program for a computer to follow, we must go through a three-phase process: *problem solving* and *implementation*  and  *maintenance* see the Fig. below

Problem solving phase Implementation Phase

Problem analysis and

Specification

General solution

(Algorithm and

flowchart)



Verification

Concrete Solution

(Program)

Testing



Maintenance Phase

Problem-Solving Phase

1. Analysis and Specification. Understand (define) the problem and what the solution must do.

2. General Solution (Algorithm). Specify the required data types and the logical sequences of steps that solve the problem.

3. Verification. Follow the steps exactly to see if the solution can really solve the problem.

Implementation Phase

1*. Concrete Solution (Program).* Translate the algorithm (the general solution) into a programming language. Choice of appropriate programming language is a crucial issue in programming. This is because the required in-built function that can make the programming easier might be missing if the language of choice is not appropriate.

2*.* Testing. Run the computer programs, and then manually check the results. If you find errors, analyze the program and the algorithm to determine the source of the errors, and then make corrections. Once a program has been written correctly, it enters a third phase: *maintenance*.

Maintenance Phase

1. This is the actual phase where the program develop will be used to solve the problem for which it was design.

2. Maintain/Modify the program to meet changing requirements or to correct any errors that show up while using it.

**Programming Language:** A programming language is a simplified form of English (with math symbols) that adheres to a strict set of grammatical rules. English is far too complicated and ambiguous for today’s computers to follow. Programming languages, because they limit vocabulary and grammar, are much simpler. Although a programming language is simple in form, it is not always easy to use. Try giving someone directions to the nearest airport using a limited vocabulary of no more than 25 words, and you begin to see the problem. Programming forces you to write very simple and exact instructions.

Translating an algorithm into a programming language is called *coding* the algorithm. The products of the translation—the code for all the algorithms in the problem—are tested by collecting them into a program and running (*executing*) the program on the computer. If the program fails to produce the desired results, the programmer must debug it—that is, determine what is wrong and then modify the program, or even one or more of the algorithms, to fix it. The combination of coding and testing the algorithms is called *implementation*.

Code is the product of translating an algorithm into a programming language. The term *code* can refer to a complete program or to any portion of a program. There is no single way to implement an algorithm. For example, an algorithm can be translated into more than one programming language. Each translation produces a different implementation. Even when two people translate an algorithm into the same programming language, they are likely to come up with different implementations. Why? Because every programming language allows the programmer some flexibility in how an algorithm is translated. Given this flexibility, people adopt their own *styles* in writing programs, just as they do in writing short stories or essays.

Some people try to speed up the programming process by going directly from the problem definition to coding the program. Developing a general solution before you write a program helps you manage the problem, keep your thoughts straight, and avoid mistakes. If you don’t take the time at the beginning to think out and polish your algorithm, you spend a lot of extra time debugging and revising your program. So *think first, plan your solution and then coding*

Documentation: this is the written text and comments that make a program easier for others to understand, use, and Modify. In addition to solving the problem, implementing the algorithm, and maintaining the program, writing documentation is an important part of the programming process. Documentation includes written explanations of the problem being solved and the organization of the solution, comments embedded within the program itself, and user manuals that describe how to use the program. Many different people are likely to work on a program over a long period of time. Each of those people must be able to read and understand the code.

ALGORITHM

An algorithm is a step-by-step procedure for calculations. More precisely, an algorithm is an effective method expressed as a finite listof well-defined instructions for solving a problem.Starting from an initial state and initial input, the instructions describe actions that, when executed, will proceed through a finite number of well-defined successive states, eventually producing "output" and terminating at a final ending state.

A flowchart is a type of diagram that represents an algorithm or process, showing the steps as boxes of various kinds, and their order by connecting them with arrows. This diagrammatic representation can give a step-by-step solution to a given problem. Process operations are represented in these boxes, and arrows connecting them represent flow of control. Arrows implied sequence of operations. Flowcharts are used in analyzing, designing, documenting or managing a process or program in various fields.

Flowchart Symbols

In general, a flowchart is a visual representation, which shows you a sequence of operations that are to be performed in order to get the solution to a problem.

Described below are standard symbols along with a visual representation right below:

* Data object - The Data object, often referred to as the I/O Shape shows the Inputs to and Outputs from a process.
* Rectangle - This is used to represent an event which is controlled within the process. Typically this will be a step or action which is taken.
* Diamond - Used to represent a decision point in the process. Typically, the statement in the symbol will require a `yes’ or `no’ response and branch to different parts of the flowchart accordingly.
* Document - The Document object is a rectangle with a wave-like base. This shape is used to represent a Document or Report in a process flow.
* Rounded box – This is used to represent an event which occurs automatically. Such an event will trigger a subsequent action, for example `receive telephone call, or describe a new state of affairs.
* Stored data - This is a general data storage object used in the process flow as opposed to data which could be also stored on a hard drive, magnetic tape, memory card, of any other storage device.
* Manual input - This object is represented by rectangle with the top sloping up from left to right. The Manual Input object signifies an action where the user is prompted for information that must be manually input into a system.
* Direct data – Direct data object in a process flow represents information stored which can be accessed directly. This object represents a computer’s hard drive.
* Circle - Used to represent a point at which the flowchart connects with another process. The name or reference for the other process should appear within the symbol.
* Internal storage – This is an object which is commonly found in programming flowcharts to illustrate the information stored in memory, as opposed to on a file.
* Predefined process – This allows you to write one subroutine and call it as often as you like from anywhere in the code.



It must also be stated that at this point in time that while learning the various symbols that are associated with flowcharts are rather important, you need to also remember that there are certain guidelines in flowcharting that deserves some respect as well. The following are some guidelines in flowcharting:

* Proper Form is Essential: In drawing a proper flowchart, all necessary requirements should be listed out in logical order.
* Clarity is Paramount: The flowchart should be clear, neat and easy to follow. There should not be any room for ambiguity in understanding the flowchart.
* Stick to the Right Direction: The usual direction of the flow of a procedure or system is from left to right or top to bottom.
* Standard for Flow Lines: Ideally just one flow line should come out from a process symbol.  While only one flow line should enter a decision symbol, around three flow lines (depending on the answer) should leave the decision symbol. Additionally, only one flow line is utilized together with a terminal symbol.
* Logic precedes everything: If you are dealing with a complex flowchart then use connector symbols to minimize the number of flow lines. Ditch the intersection of flow lines to ensure effectiveness and better communication. It is imperative that your flowchart has a logical start and finish.

Sample Problems: Suppose a programmer needs an algorithm to determine an employee’s weekly wages. The programmer begins the programming process by analyzing the problem, breaking it into manageable pieces, and developing a general solution for each piece called an algorithm The algorithm reflects what would be done by hand:

1. Look up the employee’s pay rate.

2. Determine the hours worked during the week.

3. If the number of hours worked is less than or equal to 40, multiply the hours by the pay rate to calculate regular wages.

4. If the number of hours worked is greater than 40, multiply 40 by the pay rate to calculate regular wages, and then multiply the difference between the hours worked and 40, by the overtime pay rate to calculate overtime wages.

5. Add the regular wages to the overtime wages (if any) to determine total wages for the week.

**STATEMENTS**

In computer programming a statement is the smallest standalone element of an imperative programming language. A program written in such a language is formed by a sequence of one or more statements. A statement will have internal components (e.g., expressions).

Simple statements

* Assignment statement: e.g., A:= A + 5
* Call statement: e.g. CLEARSCREEN()
* Return statement : return 5;
* goto: goto 5

Compound statement: contain more than one statements

If-statement: if A > 3 then WRITELN(A) else WRITELN("NOT YET"); end

|  |
| --- |
|  |

CONTROL STRUCTURES

Programming languages require that we use certain *control structures* to express algorithms as source code. There are four basic ways of structuring statements (instructions) in most programming languages:

* sequence, is a series of statements that are executed one after another
* selection, the conditional control structure, executes different statements depending on certain conditions
* loop, The repetitive control structure, , repeats statements while certain conditions are met and
* Subprograms,allows breaking of code into smaller units.

Each of these ways of structuring statements controls the order in which the computer executes the statements, which is why they are called control structures. Assume you’re driving a car. Going down a straight stretch of road is like following *sequence* of instructions. When you come to a T junction, on the road, you must decide which way to go and then take one or the other branch of the junction. This is what the computer does when it encounters a *selection control structure* (sometimes called a *branch* or *decision*) in a program. Sometimes you have to go around the block several times to find a place to park. The computer does the same sort of thing when it encounters a *loop* in a program. However the default setting is sequential.

Sequence selection loop

decision

Arrays

Most mathematical and computer languages have some notation for repeating or order related values. These repeated structures are often called a matrix, a dimension, or a table.

When to use arrays

It makes sense to use arrays when there are repetitive values that are related and the programmer needs to iterate through most of them. The combination of arrays and do loops in the data step lend incredible power to programming. The fact that the variables in the array do not need to be related or even contiguous makes them even more convenient to use.

Subscripts

In programming, subscript is a symbol or number used to identify an element in an array. Usually, the subscript is placed in brackets following the array name. For example, AR[5] or AR(5) identifies element number 5 in an array called AR.

If the array is multidimensional, you must specify a subscript for each dimension. For example, MD[5][3][9] or MD(5,3,9) identifies an element in a three-dimensional array called MD.

Different programming languages have different rules for specifying subscripts. For example, the BASIC language uses parentheses in place of brackets.

In word processing, a character that appears slightly below the line, as in this example: H2. A *superscript* is a character that appears slightly above the line.

An array is a collection of like variables that share a single name. The individual elements of an array are referenced by appending a *subscript*, in square brackets, behind the name. The subscript itself can be any legitimate programming expression that yields an integer value, even a general expression. Therefore, arrays in C programming language for instance may be regarded as collections of like variables. Although arrays represent one of the simplest data structures, it has wide-spread usage in embedded systems. In general we allow random access to individual array elements. On

*Array Subscripts*

When an array element is referenced, the subscript expression designates the desired element by its position in the data. The first element occupies position zero, the second position one, and so on. It follows that the last element is subscripted by [N-1] where N is the number of elements in the array. The statement

data[9] = 0;

for instance, sets the tenth element of data to zero.

If the array has two dimensions, then two subscripts are specified when referencing. As programmers we may assign logical meaning to the first and second subscripts. For example we could consider the first subscript as the row and the second as the column. Then, the statement

y = AR[3][5];

Copies the information from the 4th row 6th column into the variable y. If the array has three dimensions, then three subscripts are specified when referencing. Then, the statement

humidity[2][3][4]=100;

**Introduction To BASIC programming Language**

BASIC is a family of general-purpose, high-level programming languages whose design philosophy emphasizes ease of use; the name is an acronym from Beginner's All-purpose Symbolic Instruction Code.

The original Dartmouth BASIC was designed in 1964 by John George Kemeny and Thomas Eugene Kurtz at Dartmouth College in New Hampshire, USA to provide computer access to non-science students. At the time, nearly all use of computers required writing custom software, which was something only scientists and mathematicians tended to do. The language and its variants became widespread on microcomputers in the late 1970s and 1980s, when it was typically a standard feature, and often part of the firmware of the machine. The presence of an easy-to-learn language such as BASIC on these early personal computers allowed small business owners to develop their own custom application software, leading to widespread use of these computers in businesses that previously did not have access to computing technology.

BASIC remains popular in numerous dialects and new languages influenced by BASIC such as Microsoft Visual Basic. In 2006, 59% of developers for the .NET Framework used Visual Basic .NET as their only programming language.

|  |
| --- |
|  |

Typical BASIC keywords

Data manipulation

* LET: assigns a value (which may be the result of an [expression](http://en.wikipedia.org/wiki/Expression_%28programming%29)) to a variable.
* DATA: holds a list of values which are assigned sequentially using the READ command.

Program flow control

* IF ... THEN ... ELSE: used to perform comparisons or make decisions.
* FOR ... TO ... {STEP} ... NEXT: repeat a section of code a given number of times. A variable that acts as a counter is available within the loop.
* DO ... LOOP {WHILE} or {UNTIL}: repeat a section of code While/Until the specified condition is true. The condition may be evaluated before each iteration of the loop, or after.
* GOTO: jumps to a numbered or labeled line in the program.
* GOSUB: temporarily jumps to a numbered or labeled line, returning to the following line after encountering the RETURN Command. This is used to implement subroutines.
* ON ... GOTO/GOSUB: chooses where to jump based on the specified conditions.

Input and output

* PRINT: displays a message on the screen or other output device.
* INPUT: asks the user to enter the value of a variable. The statement may include a prompt message.

Miscellaneous

* REM: holds a programmer's comment or REMark; often used to give a title to the program and to help identify the purpose of a given section of code.

Data types and variables

Minimal versions of BASIC had only integer variables and one-letter variable names. More powerful versions had floating-point arithmetic, and variables could be labeled with names six or more characters long.

String variables are usually distinguished in many microcomputer dialects by having $ suffixed to their name, and string values are typically enclosed in quotation marks.

Arrays in BASIC could contain integers, floating point or string variables.

Some dialects of BASIC supported matrices and matrix operations, useful for the solution of sets of simultaneous linear algebraic equations. These dialects would support matrix operations such as assignment, addition, multiplication (of compatible matrix types), and evaluation of a determinant. Microcomputer dialects often lacked this data type and required a programmer to provide subroutines to carry out equivalent operations.

PROBLEM-SOLVING TECHNIQUES

You solve problems every day, often unaware of the process you are going through. In a learning environment, you usually are given most of the information you need: a clear statement of the problem, the necessary input, and the required output. In real life, the process is not always so simple. You often have to define the problem yourself and then decide what information you have to work with and what the results

Should be.

After you understand and analyze a problem, you must come up with a solution—an algorithm. Earlier we defined an algorithm as a step-by-step procedure for solving a problem in a finite amount of time with a finite amount of data. Although you work with algorithms all the time, most of your experience with them is in the context of *following* them. You follow a recipe, play a game, assemble a toy, or take medicine. In the problem-solving phase of computer programming, you will be *designing* algorithms, not following them. This means you must be conscious of the strategies you use to solve problems in order to apply them to programming problems.

1)Ask Questions

If you are given a task orally, you ask questions When? Why? Where? until you understand exactly what you have to do. If your instructions are written, you might put question marks in the margin, underline a word or a sentence, or in some other way indicate that the task is not clear. Your questions may be answered by a later paragraph, or you might have to discuss them with the person who gave you the task.

These are some of the questions you might ask in the context of programming:

• What do I have to work with—that is, what are my data?

• What do the data items look like?

• What are the operations to be performed on the data?

• How much data is there?

• How will I know when I have processed all the data?

• What should my output look like?

• What special error conditions might come up?

Look for Things That Are Familiar Never reinvent the wheel. If a solution exists, use it. If you’ve solved the same or a similar problem before, just repeat your solution. People are good at recognizing similar situations. We don’t have to learn how to go to the store to buy milk, then to buy eggs, and then to buy candy. We know that going to the store is always the same; only what we buy is different.

In programming, certain problems occur again and again in different guises. A good programmer immediately recognizes a subtask he or she has solved before and plugs in the solution. For example, finding the daily high and low temperatures is really the same problem as finding the highest and lowest grades on a test. You want the largest and smallest values in a set of numbers

2. Solve by Analogy

Often a problem reminds you of one you have seen before. You may find solving the problem at hand easier if you remember how you solved the other problem. In other words, draw an analogy between the two problems. For example, a solution to a perspective projection problem from an art class might help you figure out how to compute the distance to a landmark when you are on a cross-country hike. As you work your way through the new problem, you come across things that are different than they were in the old problem, but usually these are just details that you can deal with one at a time.

Analogy is really just a broader application of the strategy of looking for things that are familiar. When you are trying to find an algorithm for solving a problem, don’t limit yourself to computer-oriented solutions. Step back and try to get a larger view of the problem. Don’t worry if your analogy doesn’t match perfectly—the only reason for starting with an analogy is that it gives you a place to start. The best programmers are people who have broad experience in solving all kinds of problems.

3) Means-Ends Analysis

Often the beginning state and the ending state are given; the problem is to define a set of actions that can be used to get from one to the other. Suppose you want to go from Dutsinma to Kano, You know the beginning state (you are in Dutsinma) and the ending state (you want to be in Kano) The problem is how to get from one to the other. In this example, you have lots of choices. You can walk, ride a bike, drive a car. Join taxi or whatever. The method you choose depends on your circumstances. If you’re in a hurry, you’ll probably decide to fly. Once you’ve narrowed down the set of actions, you have to work out the details. It may help to establish intermediate goals that are easier to meet than the overall goal. You might decide to divide the trip into legs: campus to motor park and then park to kano Your intermediate goal is to get from Dutsinma to Kano. Now you only have to examine the means of meeting that intermediate goal. The overall strategy of means-ends analysis is to define the ends and then to analyze your means of getting between them. The process translates easily to computer programming. You begin by writing down what the input is and what the output should be. Then you consider the actions a computer can perform and choose a sequence of actions that can transform the input into the results.

4) Divide and Conquer

We often break up large problems into smaller units that are easier to handle. Cleaning the whole house may seem overwhelming; cleaning the rooms one at a time seems much more manageable. The same principle applies to programming. We break up a large problem into smaller pieces that we can solve individually.

The Building-Block Approach

Another way of attacking a large problem is to see if any solutions for smaller pieces of the problem exist. It may be possible to put some of these solutions together end to end to solve most of the big problem. This strategy is just a combination of the look-for familiar- things and divide-and-conquer approaches. You look at the big problem and see that it can be divided into smaller problems for which solutions already exist. Solving the big problem is just a matter of putting the existing solutions together, like mortaring together blocks to form a wall

Merging Solutions

Another way to combine existing solutions is to merge them on a step-by-step basis. For example, to compute the average of a list of values, we must both sum and count the values. If we already have separate solutions for summing values and for counting the number of values, we can combine them. But if we first do the summing and then do the counting, we have to read the list twice. We can save steps if we merge these two solutions: read a value and then add it to the running total and add 1 to our count before going on to the next value. Whenever the solutions to sub problems duplicate steps, think about merging them instead of joining them end to end.

Mental Blocks: The Fear of Starting

Writers are all too familiar with the experience of staring at a blank page, not knowing where to begin. Programmers have the same difficulty when they first tackle a big problem. They look at the problem and it seems overwhelming. Remember that you always have a place to begin solving any problem: Write it down on paper in your own words so that you understand it. Once you paraphrase the problem, you can focus on each of the subparts individually instead of trying to tackle the entire problem at once. This process gives you a clearer picture of the overall problem. It helps you see pieces of the problem that look familiar or that are analogous to other problems you have solved. And it pinpoints areas where something is unclear, where you need more information. As you write down a problem, you tend to group things together into small, understandable chunks of data and operations, which may be natural places to split the problem up—to divide and conquer. Your description of the problem may collect all of the information about data and results into one place for easy reference.

Rewriting the problem in your own words is a good way to focus on the subparts of the problem, one at a time, and to understand what is required for a solution.

Algorithmic Problem Solving

Coming up with an algorithm for solving a particular problem is not always cut-and dried. It fact, it is usually a trial-and-error process requiring several attempts and refinements. We test each attempt to see if it really solves the problem. If it does, fine. If it doesn’t, we try again. We typically use a combination of the techniques we’ve described to solve any nontrivial problem. Remember that the computer can only do certain things. Your primary concern, then, is how to make the computer transform, manipulate, calculate, or process the input data to produce the desired output. If you keep in mind the allowable instructions and data types in your programming language, you won’t design an algorithm that is difficult or impossible to code.