ALGORITHM

The algorithm is a sequence of instructions designed in such a way that if the instructions are executed in a specific sequence the desired results will be obtained.

An algorithm should possess the following characteristics :

1. Each and every instruction should be precise and clear

2. Each instruction should be performed a finite number of times

3. The algorithm should ultimately terminate

4. When the algorithm terminates the desired result should be obtained.

FLOWCHARTS

Symbolic representation of a solution to a given problem

Flowcharting is a tool that can help us to develop and represent graphically program logic sequence.

It also enables us to trace and detect any logical or other errors before the programs are written.

Computer professionals use two types of flowcharts viz :

- Program Flowcharts.
- System Flowcharts

-PROGRAM FLOWCHARTS.

These are used by programmers. A program flowchart shows the program structure, logic flow and operations performed.

It also forms an important part of the documentation of the system.

It broadly includes the following:

- Program Structure and Program Logic.
- Data Inputs at various stages.
- Data Processing
- Computations and Calculations.
- Conditions on which decisions are based.
- Branching & Looping Sequences.
- Results.
- Various Outputs.

The emphasis in a program flowchart is on the logic

SYSTEM FLOWCHARTS

System flowcharts are used by system analyst to show various processes, sub systems, outputs and operations on data in a system.

FLOWCHART SYMBOLS

Flowcharting has many standard symbols.

Flowcharts use boxes of different shapes to denote different types of instructions.

The actual instruction is written in the box.

These boxes are connected with solid lines which have arrowheads to indicate the direction of flow of the flowchart.

The boxes which are used in flowcharts are standardised to have specific meanings.

These flowchart symbols have been standardised by the American National Standards Institute. (ANSI).

1. TERMINAL SYMBOL:

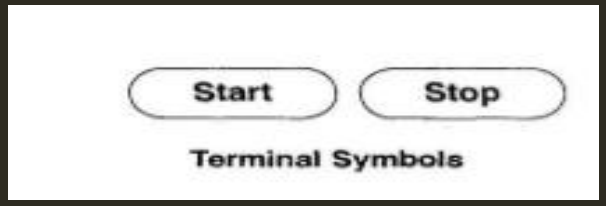
Every flowchart has a unique starting point and an ending point.

The flowchart begins at the start terminator and ends at the stop terminator.

The Starting Point is indicated with the word START inside the terminator symbol.

The Ending Point is indicated with the word STOP inside the terminator symbol.

In case a program logic involves a pause, it is also indicated with the terminal symbol

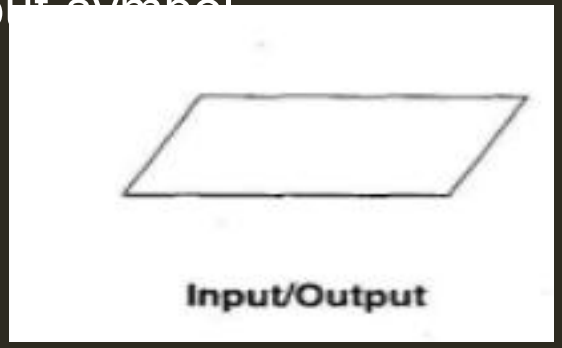


2. INPUT/OUTPUT SYMBOL :

This symbol is used to denote any input/output function in the program.

Thus if there is any input to the program via an input device, like a keyboard, tape, card reader etc. it will be indicated in the flowchart with the help of the Input/Output symbol.

Similarly, all output instructions, for output to devices like printers, plotters, magnetic tapes, disk, monitors etc. are indicated in the Input/Output symbol.



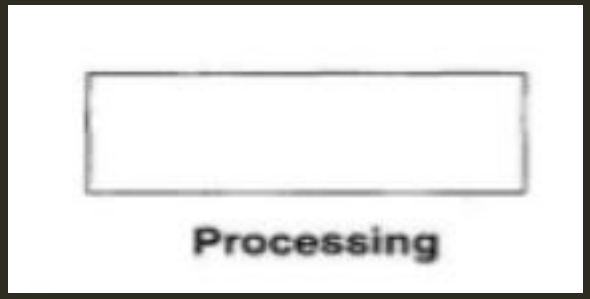
3. PROCESS SYMBOL :

A process symbol is used to represent arithmetic and data movement instructions in the flowchart.

All arithmetic processes of addition, subtraction, multiplication and division are indicated in the process symbol.

The logical process of data movement form one memory location to another is also represented in the process box.

If there are more than one process instructions to be executed sequentially, they can be placed in the same process box, one below the other in the sequence in which they are to be executed.



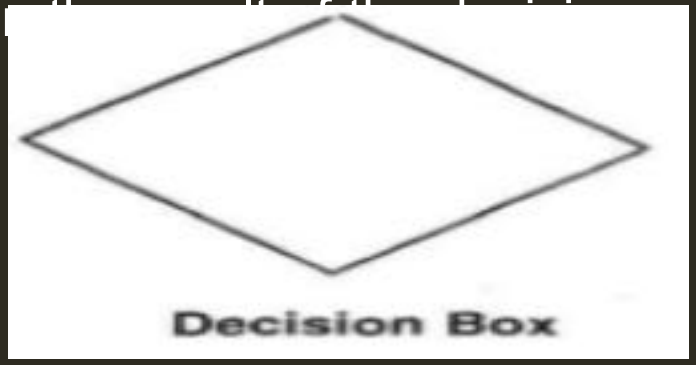
DECISION SYMBOL :

The decision symbol is used in a flowchart to indicate the point where a decision is to be made and branching done upon the result of the decision to one or more alternative paths.

The criteria for decision making is written in the decision box.

All the possible paths should be accounted for.

During execution, the appropriate path will be followed depending upo

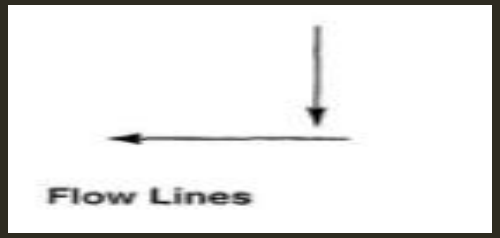


FLOWLINES :

Flowlines are solid lines with arrowheads which indicate the flow of operation.

They show the exact sequence in which the instructions are to be executed.

The normal flow of the flowchart is depicted from top to bottom and left to right.



CONNECTORS :

In situations, where the flowcharts becomes big, will result in making the flowchart difficult to understand.

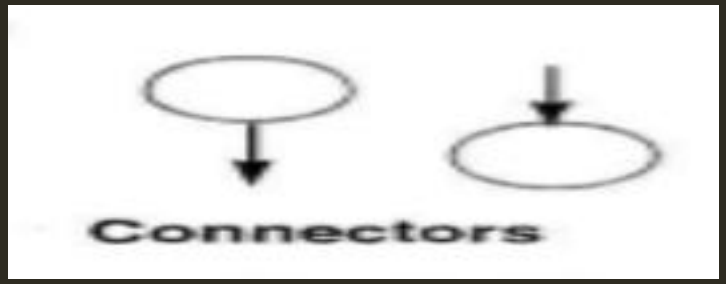
Also, the flowchart may not fit in a single page for big programs.

The connector represents entry from or exit to another part of the flowchart.

A connector symbol is indicated by a circle and a letter or a digit is placed in the circle.

This letter or digit indicates a link.

Thus a connector indicates an exit from some section in the flowchart and an entry into another section of the flowchart.



ADVANTAGES OF FLOWCHARTS

1. A flowchart being a pictorial representation of a program, makes it easier for the programmer to explain the logic of the program to others rather than a program

2. It shows the execution of logical steps without the syntax and language complexities of a program.

3. Flowcharts provide a strong documentation in the overall documentation of the software system.

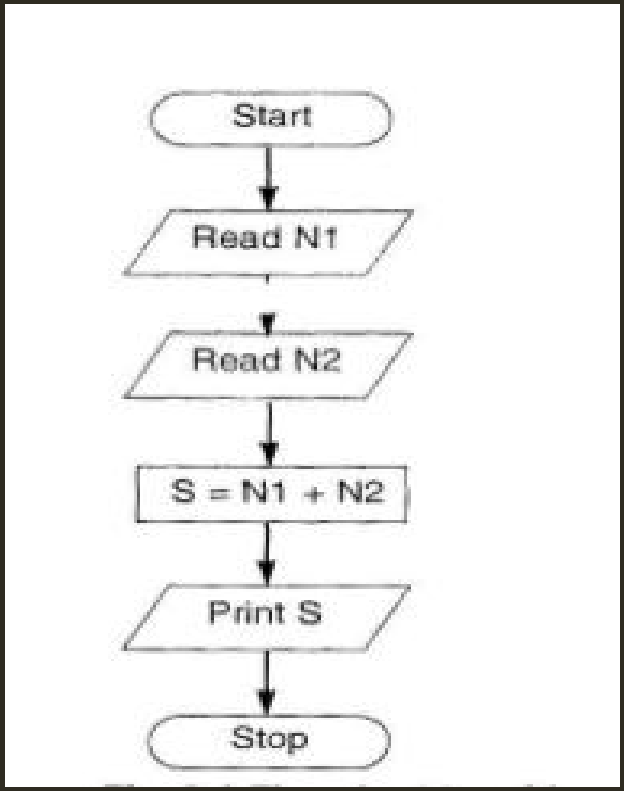
4. Once the flowchart is complete, it becomes very easy for programmers to write the program from the starting point to the ending point.

5. A flowchart is very helpful in the process of debugging a program. The bugs can be detected and corrected with the help of a flowchart in a systematic manner.

EXAMPLE : TO PREPARE A FLOWCHART TO ADD TWO NUMBERS.

The steps are :

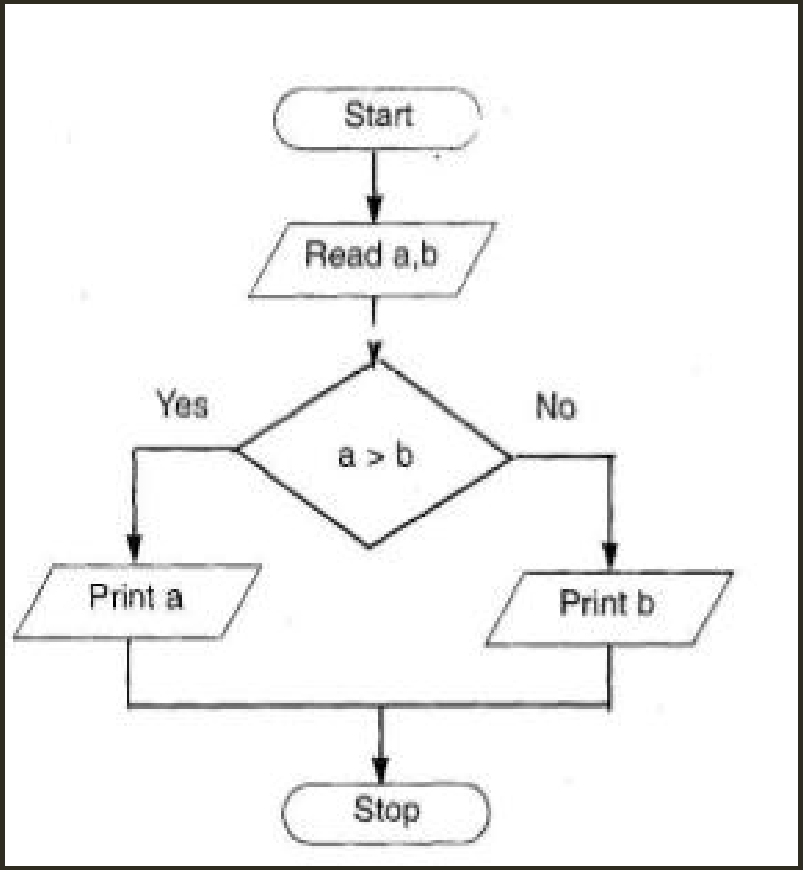
1. Start.
2. Get two numbers N1 and N2.
3. Add them.
4. Print the result.
5. Stop.



EXAMPLE : TO PREPARE A FLOWCHART TO DETERMINE THE GREATEST OF TWO NUMBERS

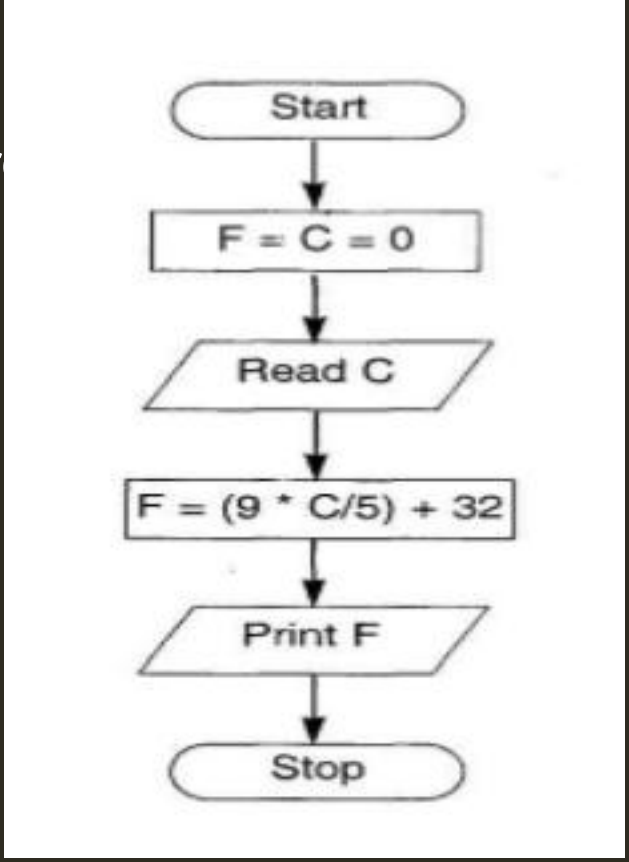
The steps are :

1. Start
2. Get two number A and B.
3. If A ı B then print A else print B.
4. Stop.



EXAMPLE: FLOWCHART FOR A PROGRAM THAT CONVERTS TEMPERATURE IN DEGREES CELSIUS TO DEGREES FAHRENHEIT.

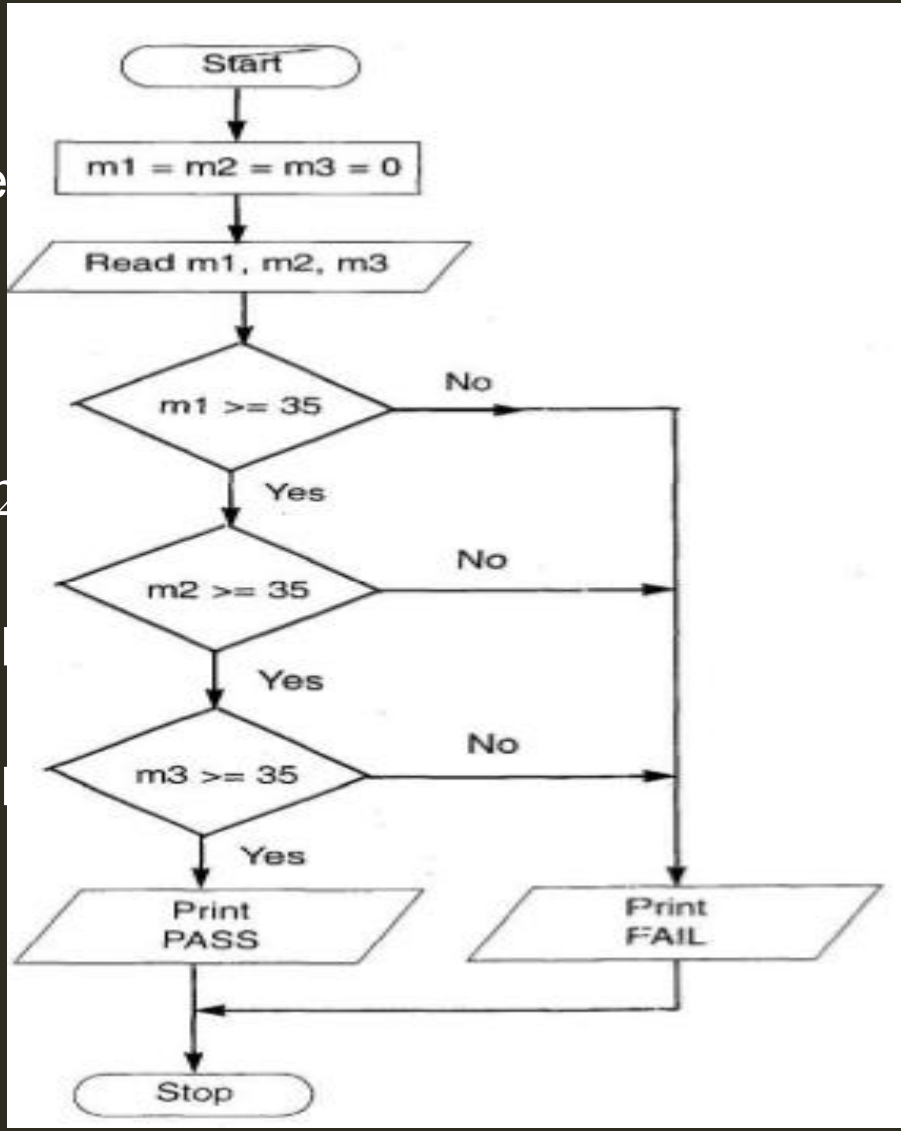
1. Start.
2. Create variables F and C (for temperatur Fahrenheit and Celsius).
3. Read degrees Celsius into C.
4. Compute the degrees Fahrenheit into F.
5. Print result (F).
6. Stop.



EXAMPLE: FLOWCHART TO GET MARKS FOR 3 SUBJECTS AND DECLARE THE RESULT.

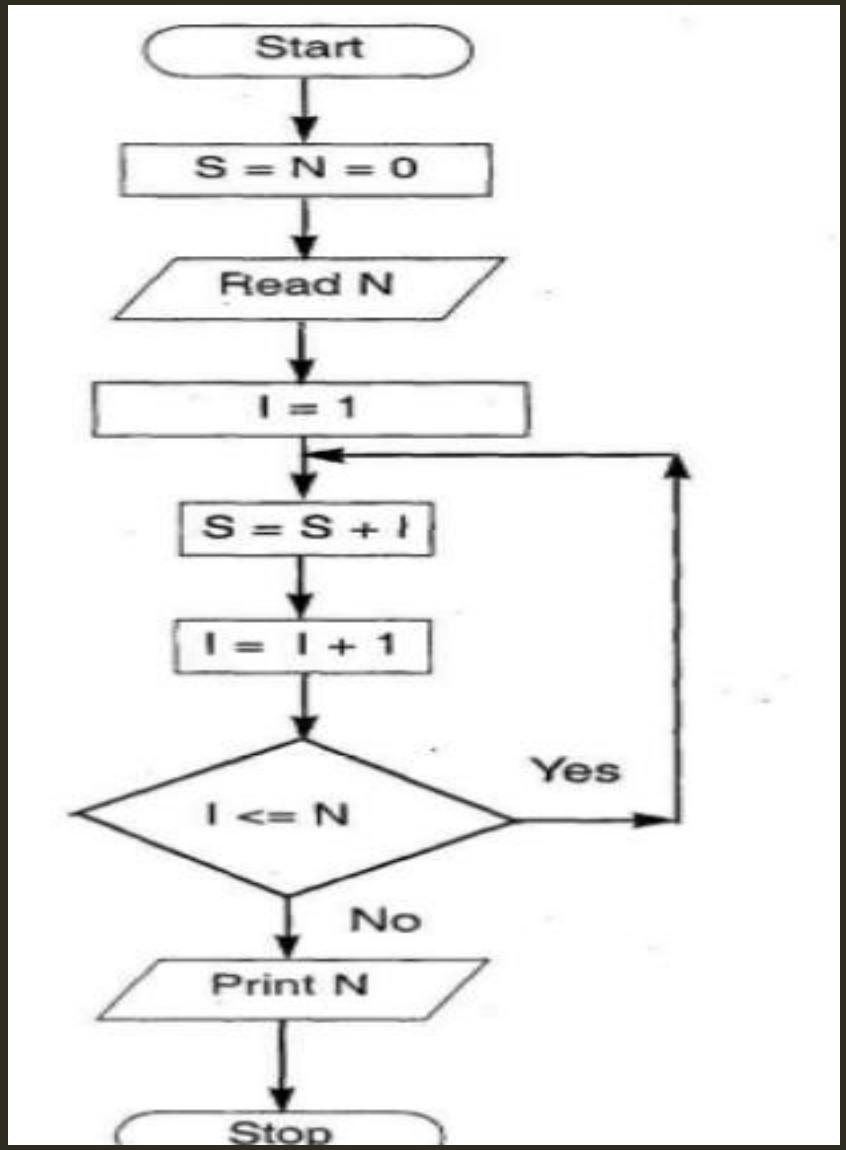
If the marks >= 35 in all the subjects the student passes else fails.

1. Start.
2. Create memvars m1, m2, m3.
3. Read marks of three subjects m1, m2 m3.
4. If m1 >= 35 goto step 5 else goto ste 7
5. If m2 >= 35 goto step 6 else goto ste 7
6. If m3 >= 35 print Pass. goto step 8
7. Print fail
8. Stop



EXAMPLE : TO FIND THE SUM OF FIRST N NUMBERS.

1. Start
2. Create memvars S , N, I
3. Read N
4. Set S (sum) to 0
5. Set counter (I) to 1.
6. S = S + I
7. Increment I by 1.
8. Check if I is less than or equal to N. If no, go to step 6.
9. Print S
10. Stop



Write the algorithm and draw flowcharts for the following :

a) Convert distance entered in Km to Metres.

b) Find the product of the just n numbers.

c) Find the sum of digits of a three digit number

Introduction to C Language

programming language developed at AT &T ‘s Bell Laboratories of USA in 1972

Designed and written by Dennis Ritchie

evolved from two previous languages, BCPL (Basic Combined Programming Language) and B.

originally implemented on a DEC PDP-11 computer in 1972.

Initially, C used widely as the development language of the UNIX OS.

Today, almost all new major OS are written in C including Windows

Getting Started With C

You will need at least:

- Computer with an OS (Linux)
- Text editor (emacs, vi,gedit)
- Compiler (gcc,cc)

All of the above suggestions are free in one way or another

- See http://www.gnu.org
- See http://www.cygwin.com

First C Program

A Simple C Program

```
1 /* A first program in C */
2         #include <stdio.h>

                          int main()
3                               {

4  printf( "Welcome to C!\n" );

          5  return 0;
          6                           }
```

```
Welcome to C!
```

Comments

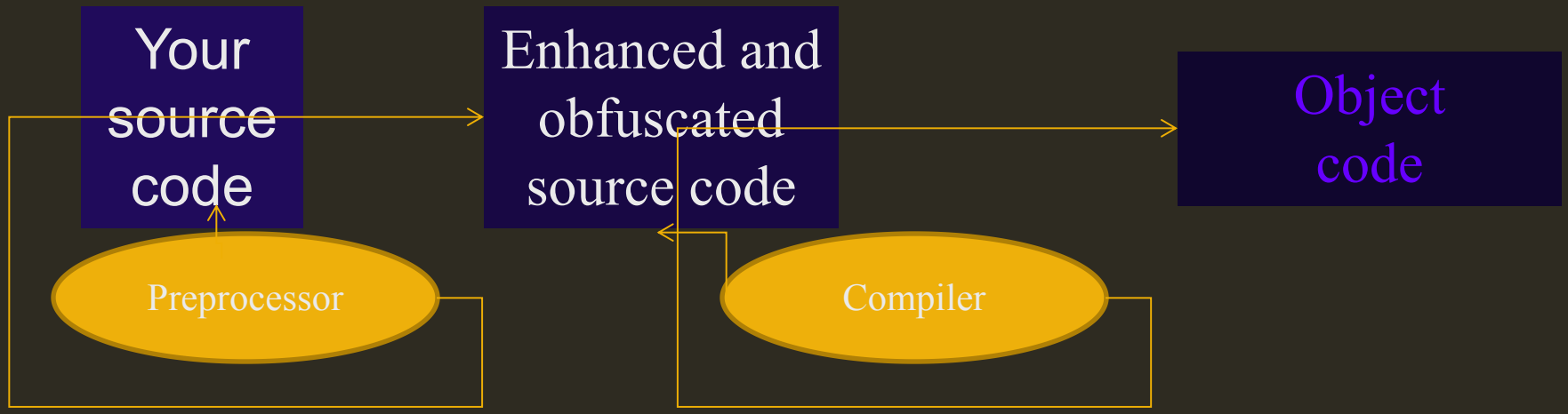
- Text surrounded by **/*** and ***/** is ignored by computer
- Used to describe program

#include <stdio.h>

- Preprocessor directive
  - Tells computer to load contents of a certain file
- **<stdio.h>** allows standard input/output operations

The preprocessor

- The preprocessor takes your source code and following certain directives that you give it tweaks it in various ways before compilation.
- A directive is given as a line of source code starting with the # symbol
- The preprocessor works in a very crude, "word-processor" way, simply cutting and pasting –
  it doesn't really know anything about C!



C Standard Header Files you may want to use

Standard Headers you should know about:

- `stdio.h` – file and console (also a file) IO: *`perror`*, *`printf`*, *`open`*, *`close`*, *`read`*, *`write`*, *`scanf`*, etc.
- `stdlib.h` - common utility functions: *`malloc`*, *`calloc`*, *`strtol`*, *`atoi`*, etc
- `string.h` - string and byte manipulation: *`strlen`*, *`strcpy`*, *`strcat`*, *`memcpy`*, *`memset`*, etc.
- `ctype.h` – character types: *`isalnum`*, *`isprint`*, *`isupport`*, *`tolower`*, etc.
- `errno.h` – defines *`errno`* used for reporting system errors
- `math.h` – math functions: *`ceil`*, *`exp`*, *`floor`*, *`sqrt`*, etc.
- `signal.h` – signal handling facility: *`raise`*, *`signal`*, etc
- `stdint.h` – standard integer: *`intN_t`*, *`uintN_t`*, etc
- `time.h` – time related facility: *`asctime`*, *`clock`*, *`time_t`*, etc.

A Simple C Program, Cont.

`int main()`

- C programs contain one or more functions, exactly one of which must be **`main`**
- Parenthesis used to indicate a function
- **`int`** means that **`main`** "returns" an integer value
- Braces (`{` and `}`) indicate a block
  - The bodies of all functions must be contained in braces

2.2 A Simple C Program: Printing a Line of Text

`Return 0;`

- A way to exit a function
- **`Return 0`**, in this case, means that the program terminated normally

Running the Program on Linux With gcc

Use emacs, vi, or some other text editor to type in and save program. Good idea to:

- Name program something meaningful
- Establish conventions for naming
- Add a .c suffix to the name

Compile program

- gcc hello.c [-o whatever]

Running on Linux

This produces the **executable** named whatever, or a.out by default.

Type executable name to run.

- Examples.
  - a.out.
  - whatever.
  - ./a.out.
  - Etc.

Note: **linker** will be required when our programs become more sophisticated – not necessary now.

Steps in learning C

CHARACTER SET (Alphabets)
CONSTANTS,VARIABLES,
KEYWORDS (Words)
INSTRUCTIONS (Sentences)
PROGRAM (Paragraphs)

The C CHARACTER SET

Alphabets (A,B…Z,a,b,…z)

Digits (0…9)

Whitespaces (blank spaces,tabs)

Special characters (, . : ; etc)

+	-	*	/	=	%	&	#
!	?	^	"	'	~	\	|
<	>	(	)	[	]	{	}
:	;	.	,	_		*(blank space)*	

Delimeters

Language pattern of C uses special symbols

:

;

()

[]

{}

#

,

TYPES OF TOKENS

Smallest unit of a Program/Statement

1. Keywords
2. Variables
3. Constants
4. Operators
5. Special characters
6. Strings

Tokens in C

Keywords

- These are reserved words of the C language.
- predefined meaning

Keywords of C

Flow control (6) – `if, else, return, switch, case, default`

Loops (5) – `for, do, while, break, continue`

Common *types* (5) – `int, float, double, char, void`

structures (3) – `struct, typedef, union`

Counting and sizing things (2) – `enum, sizeof`

Rare but still useful *types* (7) – `extern, signed, unsigned, long, short, static, const`

Evil keywords which we avoid (1) – `goto`

Wierdies (3) – `auto, register, volatile`

Identifiers

Identifiers are names given to various programming elements such as variables, functions, and arrays etc.

- An Identifier is a sequence of letters and digits, but must start with a letter.
- Both upper and lower case letters are allowed.
- Underscore (_) is treated as a letter. Identifiers are case sensitive.

Valid and invalid identifiers

y12

Sum_1

_temp

Not valid

4th

“x”

order-no

error flag

Variables in Programming

Represent storage units in a program

Named location in memory that is used to hold a value .

Used to store/retrieve data over life of program

Type of variable determines what can be placed in the storage unit

Assignment – process of placing a particular value in a variable

Variables must be *declared* before they are assigned

The value of a *variable* can be changed by program; A *constant* always has the same value

Basic C variable types

There are four basic data types in C:

- char
- int
- float
- double

Data Type	*Description*	*Typical Memory Requirements*
`int`	integer quantity	2 bytes or one word (varies from one compiler to another)
`char`	single character	1 byte
`float`	floating-point number (i.e., a number containing a decimal point and/or an exponent)	1 word (4 bytes)
`double`	double-precision floating-point number (i.e., more significant figures, and an exponent which may be larger in magnitude)	2 words (8 bytes)

Data type qualifiers

Data types can be augmented by the use of 4 data type qualifiers

long, short, signed and unsigned

Eg:- long int, signed int etc

Interpretation varies from one compiler to another

In case of an ordinary “ int “ and signed int left most bit is used to represent sign bit

Unsigned int – all bits are used to represent value (no negative numbers are allowed)

char variable type

A single character

Represents a single *byte* (8 *bits*) of storage

Internally char is just a number

Numerical value is associated with character via a *character set*.

Can be *signed* (-128 to 128)or *unsigned(0 to 255)*

ASCII character set used in ANSI C

Question: what is the difference between:

- printf(“%c”, someChar);
- printf(“%d”, someChar);

int variable type

Represents a signed integer of typically 4 or 8 bytes (32 or 64 bits) for mini computers /PCs

Precise size is machine-dependent

Question: What are the maximum and minimum sizes of the following:

- 32-bit unsigned int
- 32-bit signed int
- 64-bit signed/unsigned int

What happens if these limits are exceeded?

float and double variable types

Represent typically 32 and/or 64 bit real numbers

How these are represented internally and their precise sizes depend on the architecture. We won't obsess over this now.

Question: How large can a 64-bit float be?

Question: How many digits of precision does a 64-bit float have?

DATA TYPES IN C

32 bit gcc compiler.

Data Type	Memory (bytes)	Range	Format Specifier
short int	2	-32,768 to 32,767	%hd
unsigned short int	2	0 to 65,535	%hu
unsigned int	4	0 to 4,294,967,295	%u
int	4	-2,147,483,648 to 2,147,483,647	%d
long int	4	-2,147,483,648 to 2,147,483,647	%ld
unsigned long int	4	0 to 4,294,967,295	%lu
long long int	8	-(2^63) to (2^63)-1	%lld
unsigned long long int	8	0 to 18,446,744,073,709,551,615	%llu
signed char	1	-128 to 127	%c
unsigned char	1	0 to 255	%c
float	4		%f
double	8		%lf
long double	12		%Lf

SIZEOF( )

We can use the sizeof( ) operator to check the size of a variable.

```c
#include <stdio.h>
int main()
{
    int a = 1;
    char b ='G';
    double c = 3.14;
    printf("Hello World!\n");

    //printing the variables defined above along with their sizes
    printf("Hello! I am a character. My value is %c and "
        "my size is %lu byte.\n", b,sizeof(char));
    //can use sizeof(b) above as well

    printf("Hello! I am an integer. My value is %d and "
        "my size is %lu  bytes.\n", a,sizeof(int));
    //can use sizeof(a) above as well

    printf("Hello! I am a double floating point variable."
        " My value is %lf and my size is %lu bytes.\n",c,sizeof(double));
    //can use sizeof(c) above as well

    printf("Bye! See you soon. :)\n");

    return 0;
}
```

OUTPUT

Output:

```
Hello World!
Hello! I am a character. My value is G and my size is 1 byte.
Hello! I am an integer. My value is 1 and my size is 4  bytes.
Hello! I am a double floating point variable. My value is 3.140000 and my size i
s 8 bytes.
Bye! See you soon. :)
```

Variable Declaration

All variables must be **declared** in a C program before the first executable statement! Examples:

```
main(){
    int a, b, c;
    float d;
    /* Do something here */
}
```

```c
#include <stdio.h>
int main()
{
    // declaration and definition of variable 'a123'
    char a123 = 'a';

    // This is also both declaration and definition as 'b' is allocated
    // memory and assigned some garbage value.
    float b;

    // multiple declarations and definitions
    int _c, _d45, e;

    // Let us print a variable
    printf("%c \n", a123);

    return 0;
}
```

Variables

Naming a Variable

- Must be a valid identifier.
- Must not be a keyword
- Names are case sensitive.
- Variables are identified by only first 32 characters.
- Library commonly uses names beginning with _.
- Naming Styles: Uppercase style and Underscore style
- **`lowerLimit lower_limit`**
- **`incomeTax income_tax`**

Declaring a Variable

- Each variable used must be declared.
- A form of a declaration statement is
- **`data-type var;`**

  `data-type var1, var2,…;`
- Declaration announces the data type of a variable and allocates appropriate memory location.
- No initial value (like 0 for integers) should be assumed.

Variable assignment

After variables are declared, they must (should) be given values. This is called **assignment** and it is done with the ‘=‘ operator. Examples:

```
float a, b;
int c;
b = 2.12;
c = 200;
```