

**Operations Research File**

**Submitted to**: Dr. LN Das

**Submitted by:** Ashish Gupta

2k16/MC/023

Department of Applied Mathematics

Delhi Technological University

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| SNO. | PROGRAM | DATE | SIGNATURE |
| 1. | Simplex Method |  |  |
| 2. | Big-M Method |  |  |
| 3. | Dual Simplex Method |  |  |
| 4. | Sensitivity Analysis |  |  |
| 5. |  |  |  |
| 6. |  |  |  |
| 7. |  |  |  |
| 8. |  |  |  |
| 9. |  |  |  |
| 10. |  |  |  |

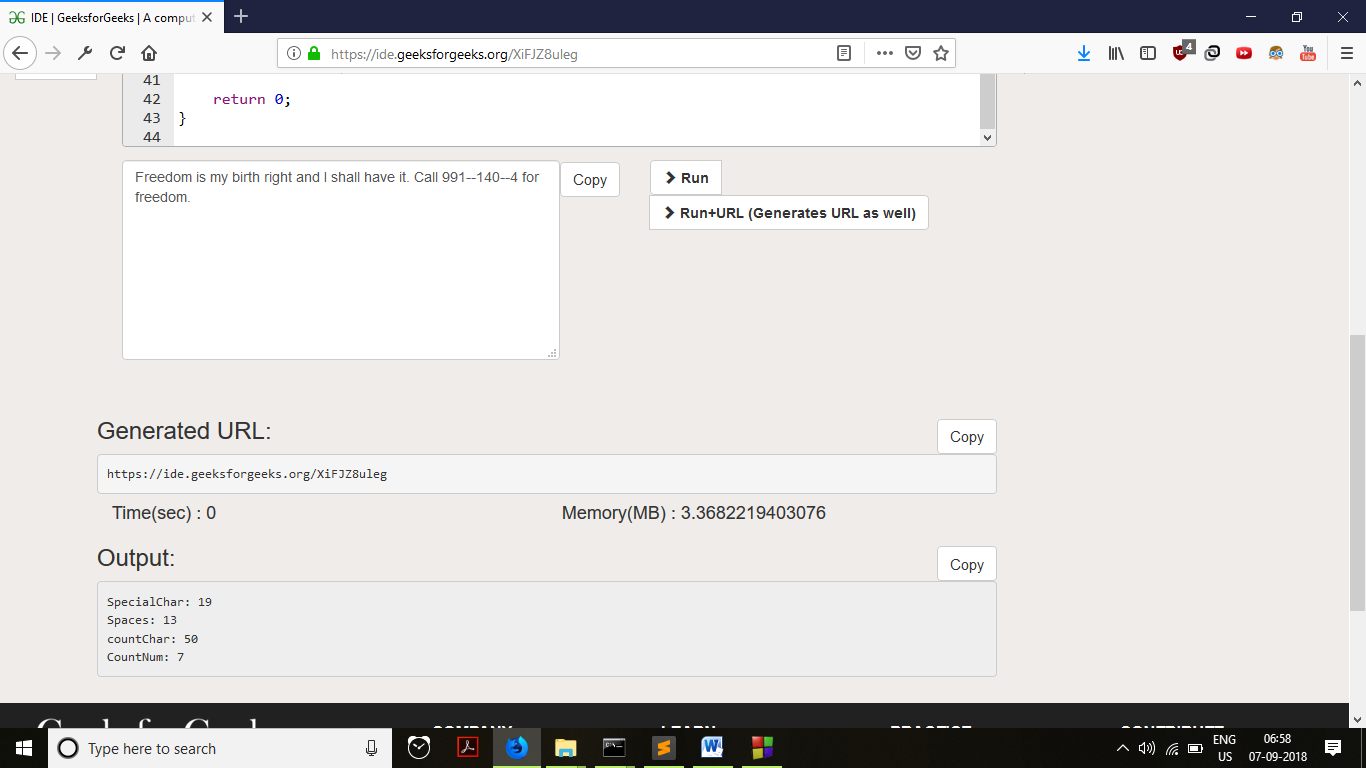
How to review what you have

**Practical 1**

**Problem Statement:**  To perform simplex method on raw data

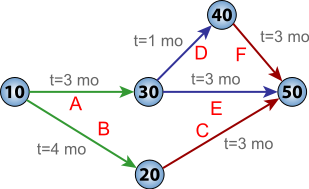
**PROGRAM:**

**Output:**



PERT Technique

The **program** (or **project**) **evaluation and review technique** (**PERT**) is a statistical tool used in [project management](https://en.wikipedia.org/wiki/Project_management), which was designed to analyze and represent the [tasks](https://en.wikipedia.org/wiki/Task_(project_management)) involved in completing a given [project](https://en.wikipedia.org/wiki/Project).



Project Evaluation and Review Techniques is commonly abbreviated to PERT. PERT is a method of analyzing the tasks involved in completing a given project, especially the time needed to complete each task, and to identify the minimum time needed to complete the total project. It incorporates uncertainty by making it possible to schedule a project while not knowing precisely the details and durations of all the activities. It is more of an event-oriented technique rather than start- and completion-oriented, and is used more in projects where time is the major factor rather than cost. It is applied to very large-scale, one-time, complex, non-routine infrastructure and Research and Development projects.

Program Evaluation Review Technique (PERT) offers a management tool, which relies on arrow and node diagrams of *activities* and *events*: arrows represent the *activities* or work necessary to reach the *events* or nodes that indicate each completed phase of the total project.

PERT and CPM are complementary tools, because "CPM employs one time estimate and one cost estimate for each activity; PERT may utilize three time estimates (optimistic, expected, and pessimistic) and no costs for each activity. Although these are distinct differences, the term PERT is applied increasingly to all critical path scheduling.

### Events and activities

In a PERT diagram, the main building block is the *event*, with connections to its known predecessor events and successor events.

* *PERT event*: a point that marks the start or completion of one or more activities. It consumes no time and uses no resources. When it marks the completion of one or more activities, it is not "reached" (does not occur) until *all* of the activities leading to that event have been completed.
* *predecessor event*: an event that immediately precedes some other event without any other events intervening. An event can have multiple predecessor events and can be the predecessor of multiple events.
* *successor event*: an event that immediately follows some other event without any other intervening events. An event can have multiple successor events and can be the successor of multiple events.

Besides events, PERT also knows activities and sub-activities:

* *PERT activity*: the actual performance of a task which consumes time and requires resources (such as labor, materials, space, machinery). It can be understood as representing the time, effort, and resources required to move from one event to another. A PERT activity cannot be performed until the predecessor event has occurred.
* *PERT sub-activity*: a PERT activity can be further decomposed into a set of sub-activities. For example, activity A1 can be decomposed into A1.1, A1.2 and A1.3. Sub-activities have all the properties of activities; in particular, a sub-activity has predecessor or successor events just like an activity. A sub-activity can be decomposed again into finer-grained sub-activities.

### Time

PERT has defined four types of time required to accomplish an activity:

* ***optimistic time***: the minimum possible time required to accomplish an activity (o) or a path (O), assuming everything proceeds better than is normally expected
* ***pessimistic time***: the maximum possible time required to accomplish an activity (p) or a path (P), assuming everything goes wrong (but excluding major catastrophes).
* *most likely time*: the best estimate of the time required to accomplish an activity (m) or a path (M), assuming everything proceeds as normal.
* ***expected time*:** the best estimate of the time required to accomplish an activity (te) or a path (TE), accounting for the fact that things don't always proceed as normal (the implication being that the expected time is the average time the task would require if the task were repeated on a number of occasions over an extended period of time).

**te** = (**o** + **4m** + **p**) ÷ **6**

T E = ∑ i = 1 n t e i {\displaystyle TE=\sum \_{i=1}^{n}te\_{i}} https://screenshotscdn.firefoxusercontent.com/images/e7af8154-d22c-41d7-800b-e17e6bc94192.png

* ***standard deviation of time*** : the variability of the time for accomplishing an activity (σte) or a path (σTE)

**σte** = (**p** - **o**) ÷ **6**

σ T E = ∑ i = 1 n σ t e i 2 {\displaystyle \sigma \_{TE}={\sqrt {\sum \_{i=1}^{n}{\sigma \_{te\_{i}}}^{2}}}} https://screenshotscdn.firefoxusercontent.com/images/2dce0762-21b9-474f-80d6-5275e128dca5.png

### Management tools

PERT supplies a number of tools for management with determination of concepts, such as:

* ***float or slack*** is a measure of the excess time and resources available to complete a task. It is the amount of time that a project task can be delayed without causing a delay in any subsequent tasks (*free float*) or the whole project (*total float*). Positive slack would indicate *ahead of schedule*; negative slack would indicate *behind schedule*; and zero slack would indicate *on schedule*.
* ***critical path***: the longest possible continuous pathway taken from the initial event to the terminal event. It determines the total calendar time required for the project; and, therefore, any time delays along the critical path will delay the reaching of the terminal event by at least the same amount.
* ***critical activity*:** An activity that has total float equal to zero. An activity with zero float is not necessarily on the critical path since its path may not be the longest.
* ***Lead time***: the time by which a *predecessor event* must be completed in order to allow sufficient time for the activities that must elapse before a specific PERT event reaches completion.
* ***lag time*:** the earliest time by which a *successor event* can follow a specific PERT event.
* ***fast tracking*:** performing more critical activities in parallel
* ***crashing critical path*:** Shortening duration of critical activities

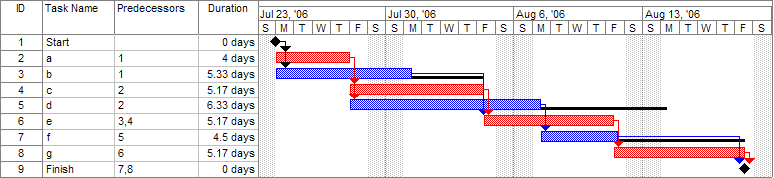
## Implementation

The first step to scheduling the project is to determine the tasks that the project requires and the order in which they must be completed. The order may be easy to record for some tasks e.g.When building a house, the land must be graded before the foundation can be laid) while difficult for others (there are two areas that need to be graded, but there are only enough bulldozers to do one). Additionally, the time estimates usually reflect the normal, non-rushed time. Many times, the time required to execute the task can be reduced for an additional cost or a reduction in the quality.

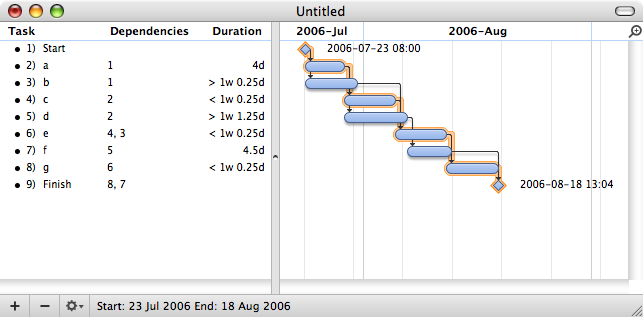
In the following example there are seven tasks, labeled *A* through *G*. Some tasks can be done concurrently (*A* and *B*) while others cannot be done until their predecessor task is complete (*C* cannot begin until *A* is complete). Additionally, each task has three time estimates: the optimistic time estimate (*o*), the most likely or normal time estimate (*m*), and the pessimistic time estimate (*p*). The expected time (*te*) is computed using the formula (*o* + 4*m* + *p*) ÷ 6.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Activity | Predecessor | Time estimates | | | Expected time |
| **Opt. (*o*)** | **Normal (*m*)** | **Pess. (*p*)** |
| *A* | — | 2 | 4 | 6 | 4.00 |
| *B* | — | 3 | 5 | 9 | 5.33 |
| *C* | *A* | 4 | 5 | 7 | 5.17 |
| *D* | *A* | 4 | 6 | 10 | 6.33 |
| *E* | *B*, *C* | 4 | 5 | 7 | 5.17 |
| *F* | *D* | 3 | 4 | 8 | 4.50 |
| *G* | *E* | 3 | 5 | 8 | 5.17 |

Once this step is complete, one can draw a Gantt chart or a network diagram.

[](https://en.wikipedia.org/wiki/File:Pert_example_gantt_chart.gif)

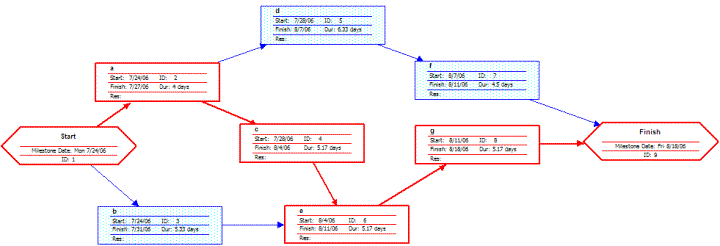
A Gantt chart created using [Microsoft Project](https://en.wikipedia.org/wiki/Microsoft_Project) (MSP). Note (1) the [critical path](https://en.wikipedia.org/wiki/Critical_path_method) is in red, (2) the [slack](https://en.wikipedia.org/wiki/Float_(project_management)) is the black lines connected to non-critical activities, (3) since Saturday and Sunday are not work days and are thus excluded from the schedule, some bars on the Gantt chart are longer if they cut through a weekend.

[](https://en.wikipedia.org/wiki/File:Pert_example_gantt_chart.png)

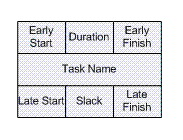
note (1) the [critical path](https://en.wikipedia.org/wiki/Critical_path_method) is highlighted, (2) the [slack](https://en.wikipedia.org/wiki/Float_(project_management)) is not specifically indicated on task 5 (d), though it can be observed on tasks 3 and 7 (b and f), (3) since weekends are indicated by a thin vertical line, and take up no additional space on the work calendar, bars on the Gantt chart are not longer or shorter when they do or don't carry over a weekend.

### Next step, creating network diagram by hand or by using diagram software

A network diagram can be created by hand or by using diagram software. There are two types of network diagrams, activity on arrow (AOA) and activity on node (AON). Activity on node diagrams are generally easier to create and interpret. To create an AON diagram, it is recommended (but not required) to start with a node named *start*. This "activity" has a duration of zero (0). Then you draw each activity that does not have a predecessor activity (*a* and *b* in this example) and connect them with an arrow from start to each node. Next, since both *c* and *d* list *a* as a predecessor activity, their nodes are drawn with arrows coming from *a*. Activity *e* is listed with *b* and *c* as predecessor activities, so node *e* is drawn with arrows coming from both *b* and *c*, signifying that *e* cannot begin until both *b* and *c* have been completed. Activity *f* has *d* as a predecessor activity, so an arrow is drawn connecting the activities. Likewise, an arrow is drawn from *e* to *g*. Since there are no activities that come after *f* or *g*, it is recommended (but again not required) to connect them to a node labeled *finish*.

[](https://en.wikipedia.org/wiki/File:Pert_example_network_diagram.gif)

Note the critical path is in red.

[](https://en.wikipedia.org/wiki/File:Pert_example_node_legend.GIF)

A node like this one can be used to display the activity name, duration, ES, EF, LS, LF, and slack.

By itself, the network diagram pictured above does not give much more information than a Gantt chart; however, it can be expanded to display more information. The most common information shown is:

1. The activity name
2. The expected duration time
3. The early start time (ES)
4. The early finish time (EF)
5. The late start time (LS)
6. The late finish time (LF)
7. The slack

In order to determine this information it is assumed that the activities and normal duration times are given. The first step is to determine the ES and EF. The ES is defined as the maximum EF of all predecessor activities, unless the activity in question is the first activity, for which the ES is zero (0). The EF is the ES plus the task duration (EF = ES + duration).

* The ES for *start* is zero since it is the first activity. Since the duration is zero, the EF is also zero. This EF is used as the ES for *a* and *b*.
* The ES for *a* is zero. The duration (4 work days) is added to the ES to get an EF of four. This EF is used as the ES for *c* and *d*.
* The ES for *b* is zero. The duration (5.33 work days) is added to the ES to get an EF of 5.33.
* The ES for *c* is four. The duration (5.17 work days) is added to the ES to get an EF of 9.17.
* The ES for *d* is four. The duration (6.33 work days) is added to the ES to get an EF of 10.33. This EF is used as the ES for *f*.
* The ES for *e* is the greatest EF of its predecessor activities (*b* and *c*). Since *b* has an EF of 5.33 and *c* has an EF of 9.17, the ES of *e* is 9.17. The duration (5.17 work days) is added to the ES to get an EF of 14.34. This EF is used as the ES for *g*.
* The ES for *f* is 10.33. The duration (4.5 work days) is added to the ES to get an EF of 14.83.
* The ES for *g* is 14.34. The duration (5.17 work days) is added to the ES to get an EF of 19.51.
* The ES for *finish* is the greatest EF of its predecessor activities (*f* and *g*). Since *f* has an EF of 14.83 and *g* has an EF of 19.51, the ES of *finish* is 19.51. *Finish* is a milestone (and therefore has a duration of zero), so the EF is also 19.51.

Barring any unforeseen events, the project should take 19.51 work days to complete. The next step is to determine the late start (LS) and late finish (LF) of each activity. This will eventually show if there are activities that have slack. The LF is defined as the minimum LS of all successor activities, unless the activity is the last activity, for which the LF equals the EF. The LS is the LF minus the task duration (LS = LF − duration).

* The LF for *finish* is equal to the EF (19.51 work days) since it is the last activity in the project. Since the duration is zero, the LS is also 19.51 work days. This will be used as the LF for *f* and *g*.
* The LF for *g* is 19.51 work days. The duration (5.17 work days) is subtracted from the LF to get an LS of 14.34 work days. This will be used as the LF for *e*.
* The LF for *f* is 19.51 work days. The duration (4.5 work days) is subtracted from the LF to get an LS of 15.01 work days. This will be used as the LF for *d*.
* The LF for *e* is 14.34 work days. The duration (5.17 work days) is subtracted from the LF to get an LS of 9.17 work days. This will be used as the LF for *b* and *c*.
* The LF for *d* is 15.01 work days. The duration (6.33 work days) is subtracted from the LF to get an LS of 8.68 work days.
* The LF for *c* is 9.17 work days. The duration (5.17 work days) is subtracted from the LF to get an LS of 4 work days.
* The LF for *b* is 9.17 work days. The duration (5.33 work days) is subtracted from the LF to get an LS of 3.84 work days.
* The LF for *a* is the minimum LS of its successor activities. Since *c* has an LS of 4 work days and *d* has an LS of 8.68 work days, the LF for *a* is 4 work days. The duration (4 work days) is subtracted from the LF to get an LS of 0 work days.
* The LF for *start* is the minimum LS of its successor activities. Since *a* has an LS of 0 work days and *b* has an LS of 3.84 work days, the LS is 0 work days.

### Next step, determination of critical path and possible slack

The next step is to determine the critical path and if any activities have slack. The critical path is the path that takes the **longest** to complete. To determine the path times, add the task durations for all available paths. Activities that have slack can be delayed without changing the overall time of the project. Slack is computed in one of two ways, slack = LF − EF *or* slack = LS − ES. Activities that are on the critical path have a slack of zero (0).

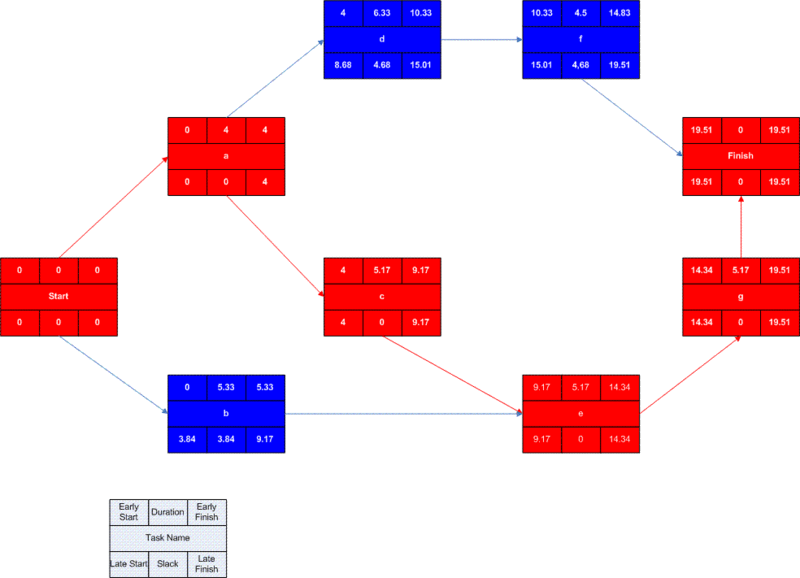
* The duration of path *adf* is 14.83 work days.
* The duration of path *aceg* is 19.51 work days.
* The duration of path *beg* is 15.67 work days.

The critical path is *aceg* and the critical time is 19.51 work days. It is important to note that there can be more than one critical path (in a project more complex than this example) or that the critical path can change. For example, let's say that activities *d* and *f* take their pessimistic (b) times to complete instead of their expected (TE) times. The critical path is now *adf* and the critical time is 22 work days. On the other hand, if activity *c* can be reduced to one work day, the path time for *aceg* is reduced to 15.34 work days, which is slightly less than the time of the new critical path, *beg* (15.67 work days).

Assuming these scenarios do not happen, the slack for each activity can now be determined.

* *Start* and *finish* are milestones and by definition have no duration, therefore they can have no slack (0 work days).
* The activities on the critical path by definition have a slack of zero; however, it is always a good idea to check the math anyway when drawing by hand.
  + LFa – EFa = 4 − 4 = 0
  + LFc – EFc = 9.17 − 9.17 = 0
  + LFe – EFe = 14.34 − 14.34 = 0
  + LFg – EFg = 19.51 − 19.51 = 0
* Activity *b* has an LF of 9.17 and an EF of 5.33, so the slack is 3.84 work days.
* Activity *d* has an LF of 15.01 and an EF of 10.33, so the slack is 4.68 work days.
* Activity *f* has an LF of 19.51 and an EF of 14.83, so the slack is 4.68 work days.

Therefore, activity *b* can be delayed almost 4 work days without delaying the project. Likewise, activity *d* **or** activity *f* can be delayed 4.68 work days without delaying the project (alternatively, *d* and *f* can be delayed 2.34 work days each).

[](https://en.wikipedia.org/wiki/File:Pert_example_network_diagram_visio.gif)

### Advantages

* PERT chart explicitly defines and makes visible [dependencies](https://en.wikipedia.org/wiki/Dependency_(project_management)) (precedence relationships) between the work breakdown structure (commonly [WBS](https://en.wikipedia.org/wiki/Work_breakdown_structure)) elements.
* PERT facilitates identification of the critical path and makes this visible.
* PERT facilitates identification of early start, late start, and slack for each activity.
* PERT provides for potentially reduced project duration due to better understanding of dependencies leading to improved overlapping of activities and tasks where feasible.
* The large amount of project data can be organized and presented in diagram for use in decision making.
* PERT can provide a probability of completing before a given time.

### Disadvantages

* There can be potentially hundreds or thousands of activities and individual dependency relationships.
* PERT is not easily scalable for smaller projects.
* The network charts tend to be large and unwieldy, requiring several pages to print and requiring specially-sized paper.
* The lack of a timeframe on most PERT/CPM charts makes it harder to show status, although colours can help, *e.g.*, specific colour for completed nodes.

Case Study:

**How a computer does compile tasks**

**How does a computer converts it to machine language code?**

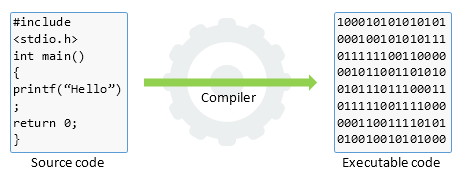
**How machine language code is executed using registers?**

**How those registers are used to display the results on the screen?**

# Step 1: Compiling a program (TORA OR C++ OR C )

## What is meant by Compilation?

The process of translating source code written in [high level](https://codeforwin.org/2017/05/high-level-languages-advantages-disadvantages.html) to [low level](https://codeforwin.org/2017/05/low-level-languages-advantages-disadvantages.html) machine code is called as Compilation. The compilation is done by a special software known as [compiler](https://codeforwin.org/2017/05/compiler-and-its-need.html). The compiler checks source code for any syntactical or structural errors and generates object code with extension **.obj** (in Windows) or **.o** (in Linux) if source code is error free.

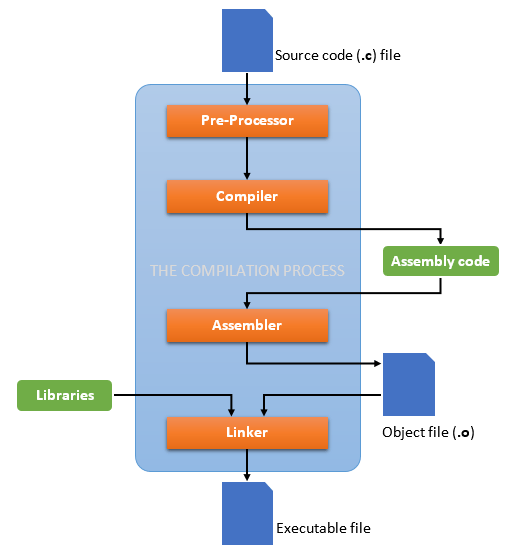


SOURCE CODE AND EXECUTABLE CODE

## The compilation

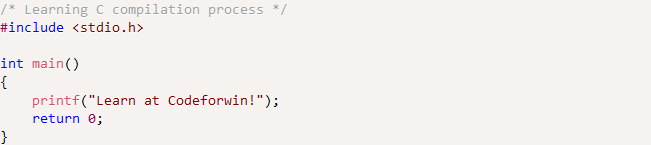
The entire C compilation is broken to four stages.

The below image describes the entire C compilation process.



The C compilation process

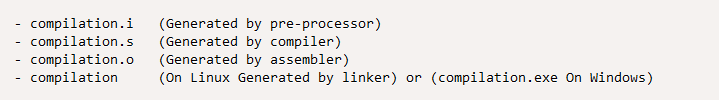
To take a deep dive inside the C compilation process let’s compile a C program. Write or copy below C program and save it as **compilation.c.**



To compile the above program open command prompt and hit below command.

https://screenshotscdn.firefoxusercontent.com/images/e32dc304-98db-4f8b-a5c9-c630ee446542.png

The **-save-temps** option will preserve and save all temporary files created during the C compilation. It will generate four files in the same directory namely.



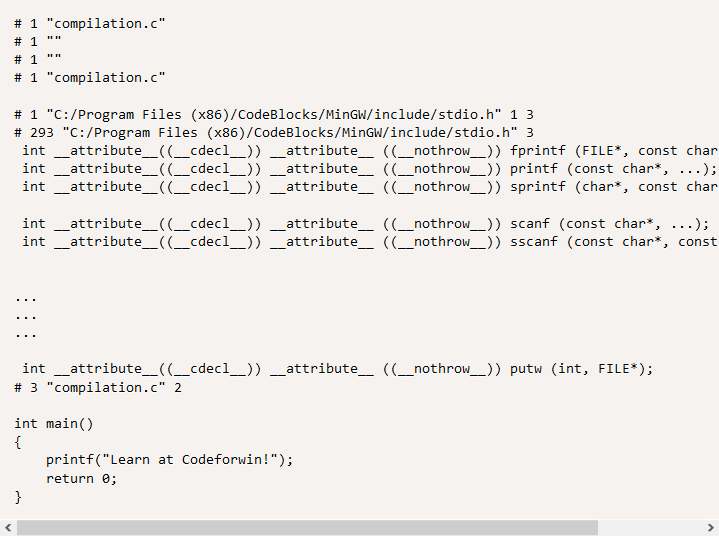
### Pre-processing of source file

The C compilation begins with pre-processing of source file. **Pre-processor** is a small software that accepts C source file and performs below tasks.

* Remove comments from the source code.
* Macro expansion.
* Expansion of included header files.

After pre-processing it generates a temporary file with .i extension. Since, it inserts contents of header files to our source code file. Pre-processor generated file is larger than the original source file.

To view contents of the pre-processed file open <file-name>.i in your favourite text editor. As in our case below is an extract of compilation.i file.

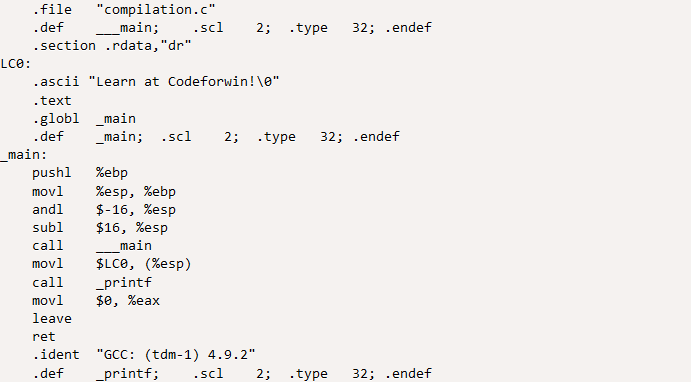


### Compilation of pre-processed file

In next phase of C compilation the compiler comes in action. It accepts temporary pre-processed <file-name>.i file generated by the pre-processor and performs following tasks.

* Check C program for syntax errors.
* Translate the file into intermediate code i.e. in assembly language.
* Optionally optimize the translated code for better performance.

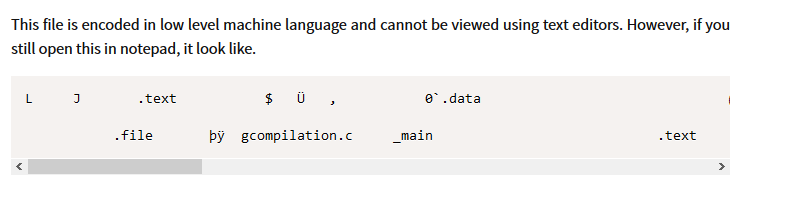
After compiling it generates an intermediate code in assembly language as <file-name.s> file. It is assembly version of our source code.  
Let us look into compilation.s file.



### Assembling of compiled source code

Moving on to the next phase of compilation. Assembler accepts the compiled source code (**compilation.s**) and translates to low level machine code. After successful assembling it generates <file-name.o> (in Linux) or <file-name.obj> (in Windows) file known as object file. In our case it generates the compilation.o file.

This file is encoded in low level machine language and cannot be viewed using text editors. However, if you still open this in notepad, it look like.

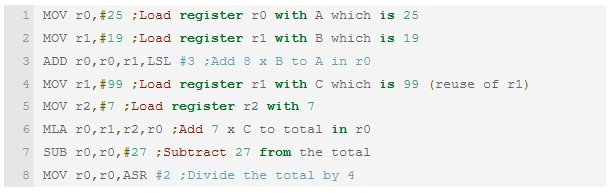


### Linking of object files

Finally, the linker comes in action and performs the final task of compilation process. It accepts the intermediate file <file-name.o> generated by the assembler. It links all the function calls with their original definition. Which means the function printf() gets linked to its original definition.

Linker generates the final executable file (.exe in windows)

### Registers and memory management



* Could do things like display graphics on the screen or read input from the keyboard.
* With machine language we use very simple instructions to tell the computer to:
* Load and save values from memory.
* Do arithmetic and logical operations.
* Very simple control structures (pretty much limited to something like ‘GOTO’)

The answer is that memory is not just a storage areas in a computer. Different parts of the memory are mapped to the hardware in various ways. The image on the screen corresponds to a certain region of memory where the values determine the colour of each pixel.

For input, in some cases you can read values from special locations in memory, which are updated by the hardware. In other cases the processor will be driven by ‘interrupts’. Basically, part of the hardware sends a signal to the processor. It interrupts what it is doing and jumps to a special area of memory where code is stored to handle the signal. When it’s dealt with, the processor resumes executing your code.

The computer ‘knows how to execute the instructions’ because the hardware is set up to respond to certain patterns of 0s and 1s in a certain way. The logic gates on the chip are arranged such that when various patterns of input are read, it will perform the intended operations. With machine code you are getting as close to the physical system as possible.

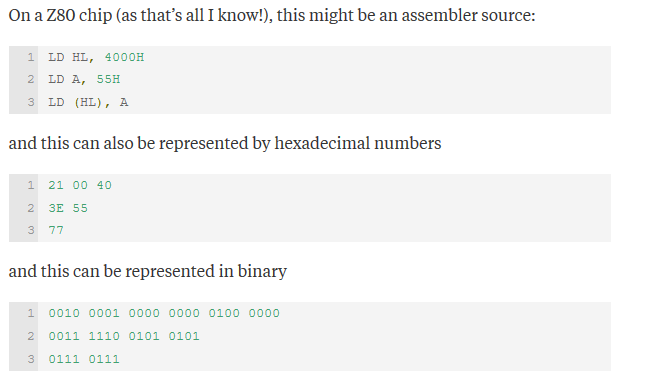
Now, how does the computer know what to do with the 1’s and 0’s? Inside the computer is what is known as the CPU (Central Processing Unit). This has an instruction set in side that understands these 1’s and 0’s and takes actions that these 1’s and 0’s tell it to do. For instance a certain value, let’s say 64 tells the computer to load the accumulator with a value. The value is the next number that is sent to it. What’s the accumulator? It is a memory location where just about everything is done. There are other memory locations that have their jobs as well but the accumulator is the single most important one; the others just help.

Every number can be converted into 1’s and 0’s just as every group of 1’s and 0’s can be converted into numbers that humans can recognize. The 1’s and 0’s are called binary and it only has those two values. 64 would look like 01000000 in binary. If your interested in binary there are books on the subject.

The processor is connected to memory. Computer memory can be thought of as a series of cells which can contain a value. Each cell has an address (0, 1, 2, 3, …). The processor can ask what value is stored in a certain location in memory, or set a certain location to a value. The programs themselves are also stored in memory and there is a special register on the processor to record where the next instruction is coming from (the Program Counter).

The x86 series in many desktop machines is much more complicated.

Example:



Step 2: How does A Display Work?



Photo: Small LCDs like this one have been widely used in calculators and digital watches since the 1970s, but they were relatively expensive in those days and produced only black-and-white (actually, dark-blueish and white) images. During the 1980s and 1990s, manufacturers figured out how to make larger color screens at relatively affordable prices. That was when the market for LCD TVs and color laptop computers really took off.

# Computer Displays

## How does a Computer screen make its picture?



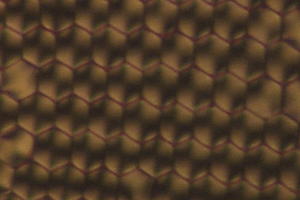
Photo: This [iPod](https://www.explainthatstuff.com/how-mp3players-work.html) screen is another example of LCD technology. Its pixels are colored black and they're either on or off, so the display is black-and-white. In an LCD TV screen, much smaller pixels colored red, blue, or green make a brightly colored moving picture.

For many people, the most attractive thing about LCD TVs is not the way they make a picture but their flat, compact screen. Unlike an old-style TV, an LCD screen is flat enough to hang on your wall. That's because it generates its picture in an entirely different way.

You probably know that an old-style [cathode-ray tube (CRT) television](https://www.explainthatstuff.com/television.html) makes a picture using three electron guns. Think of them as three very fast, very precise paintbrushes that dance back and forth, painting a moving image on the back of the screen that you can watch when you sit in front of it.

Flatscreen LCD and plasma screens work in a completely different way. If you sit up close to a flatscreen TV, you'll notice that the picture is made from millions of tiny blocks called pixels (picture elements). Each one of these is effectively a separate red, blue, or green light that can be switched on or off very rapidly to make the moving color picture. The pixels are controlled in completely different ways in plasma and LCD screens. In a [plasma](https://www.explainthatstuff.com/plasmatv.html) screen, each pixel is a tiny fluorescent lamp switched on or off [electronically](https://www.explainthatstuff.com/electronics.html). In an LCD television, the pixels are switched on or off electronically using liquid crystals to rotate polarized light. That's not as complex as it sounds! To understand what's going on, first we need to understand what liquid crystals are; then we need to look more closely at light and how it travels.

## What are liquid crystals?



* Photo: Liquid crystals dried and viewed through polarized light. You can see they have a much more regular structure than an ordinary liquid. Photo from research by David Weitz courtesy of NASA Marshall Space Flight Center (NASA-MSFC).

We're used to the idea that a given substance can be in one of three states: solid, liquid, or gas—we call them [states of matter](https://www.explainthatstuff.com/states-of-matter.html)—and up until the late 19th century, scientists thought that was the end of the story. Then, in 1888, an Austrian chemist named Friedrich Reinitzer (1857–1927) discovered liquid crystals, which are another state entirely, somewhere in between liquids and solids. Liquid crystals might have lingered in obscurity but for the fact that they turned out to have some very useful properties.

Solids are frozen lumps of matter that stay put all by themselves, often with their [atoms](https://www.explainthatstuff.com/atoms.html) packed in a neat, regular arrangement called a crystal (or crystalline lattice). Liquids lack the order of solids and, though they stay put if you keep them in a container, they flow relatively easily when you pour them out. Now imagine a substance with some of the order of a solid and some of the fluidity of a liquid. What you have is a liquid crystal—a kind of halfway house in between. At any given moment, liquid crystals can be in one of several possible "substates" (phases) somewhere in a limbo-land between solid and liquid. The two most important liquid crystal phases are called nematic and smectic:



* When they're in the nematic phase, liquid crystals are a bit like a liquid: their molecules can move around and shuffle past one another, but they all point in broadly the same direction. They're a bit like matches in a matchbox: you can shake them and move them about but they all keep pointing the same way.
* If you cool liquid crystals, they shift over to the smectic phase. Now the molecules form into layers that can slide past one another relatively easily. The molecules in a given layer can move about within it, but they can't and don't move into the other layers (a bit like people working for different companies on particular floors of an office block). There are actually several different smectic "subphases," but we won't go into them.

## What is polarized light?

Nematic liquid crystals have a really neat party trick. They can adopt a twisted-up structure and, when you apply [electricity](https://www.explainthatstuff.com/electricity.html) to them, they straighten out again. That may not sound much of a trick, but it's the key to how LCD displays turn pixels on and off. To understand how liquid crystals can control pixels, we need to know about polarized light.

[Light](https://www.explainthatstuff.com/light.html) is a mysterious thing. Sometimes it behaves like a stream of particles—like a constant barrage of microscopic cannonballs carrying [energy](https://www.explainthatstuff.com/energy.html) we can see, through the air, at extremely high speed. Other times, light behaves more like waves on the sea. Instead of water moving up and down, light is a wave pattern of electrical and [magnetic](https://www.explainthatstuff.com/magnetism.html) energy vibrating through space.



* Photo: A trick of the polarized light: rotate one pair of polarizing sunglasses past another and you can block out virtually all the light that normally passes through.



* Photo: A less well known trick of polarized light: it makes crystals gleam with amazing spectral colors due to a phenomenon called [pleochroism](http://edafologia.ugr.es/OptMine/ppl/pleow.htm). Photo of protein and virus crystals, many of which were grown in space. Credit: Dr. Alex McPherson, University of California, Irvine. Photo courtesy of NASA Marshall Space Flight Center (NASA-MSFC).

When sunlight streams down from the sky, the light waves are all mixed up and vibrating in every possible direction. But if we put a filter in the way, with a grid of lines arranged vertically like the openings in prison bars (only much closer together), we can block out all the light waves except the ones vibrating vertically (the only light waves that can get through vertical bars). Since we block off much of the original sunlight, our filter effectively dims the light. This is how polarizing sunglasses work: they cut out all but the sunlight vibrating in one direction or plane. Light filtered in this way is called polarized or plane-polarized light (because it can travel in only one plane).

If you have two pairs of polarizing sunglasses (and it won't work with ordinary sunglasses), you can do a clever trick. If you put one pair directly in front of the other, you should still be able to see through. But if you slowly rotate one pair, and keep the other pair in the same place, you will see the light coming through gradually getting darker. When the two pairs of sunglasses are at 90 degrees to each other, you won't be able to see through them at all. The first pair of sunglasses blocks off all the light waves except ones vibrating vertically. The second pair of sunglasses works in exactly the same way as the first pair. If both pairs of glasses are pointing in the same direction, that's fine—light waves vibrating vertically can still get through both. But if we turn the second pair of glasses through 90 degrees, the light waves that made it through the first pair of glasses can no longer make it through the second pair. No light at all can get through two polarizing filters that are at 90 degrees to one another.

## How LCD televisions use liquid crystals and polarized light



Photo: Prove to yourself that an LCD display uses polarized light. Simply put on a pair of polarizing sunglasses and rotate your head (or the display). You'll see the display at its brightest at one angle and at its darkest at exactly 90 degrees to that angle.

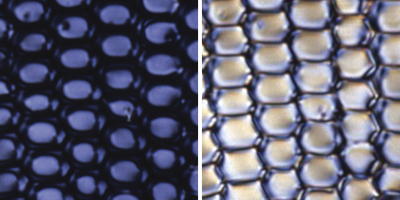


Photo: How liquid crystals switch light on and off. In one orientation, polarized light cannot pass through the crystals so they appear dark (left side photo). In a different orientation, polarized light passes through okay so the crystals appear bright (right side photo). We can make the crystals change orientation—and switch their pixels on and off—simply by applying an electric field. Photo from liquid crystal research by David Weitz courtesy of NASA Marshall Space Flight Center (NASA-MSFC).

An LCD TV screen uses the sunglasses trick to switch its colored pixels on or off. At the back of the screen, there's a large bright light that shines out toward the viewer. In front of this, there are the millions of pixels, each one made up of smaller areas called sub-pixels that are colored red, blue, or green. Each pixel has a polarizing glass filter behind it and another one in front of it at 90 degrees. That means the pixel normally looks dark. In between the two polarizing filters there's a tiny twisted, nematic liquid crystal that can be switched on or off (twisted or untwisted) electronically. When it's switched off, it rotates the light passing through it through 90 degrees, effectively allowing light to flow through the two polarizing filters and making the pixel look bright. When it's switched on, it doesn't rotate the light, which is blocked by one of the polarizers, and the pixel looks dark. Each pixel is controlled by a separate [transistor](https://www.explainthatstuff.com/howtransistorswork.html) (a tiny electronic component) that can switch it on or off many times each second.

