# Project – Air Traffic Monitoring and Control System

**Deadline:** Friday November 30th by midnight
**Type:** Group of 2 students maximum
**Weight:** This programming project is worth 15% of your final grade
**Submission:** Must be on Moodle

**Policy for late hand-in:** After the deadline, the project will not be accepted.

**Marking Scheme:**
Code correctness (75%)
Code output format, clarity, completeness, and accuracy (10%)
Report (15%)

---

## A. Goal

This project involves all or part of the design, implementation, testing, and analysis of a simplified air traffic monitoring and control system (ATC). The system is a variation of the simplified ATC described in the next section. A standard computer system with an accessible clock is assumed. The project specification is intentionally inexplicit in some parts such as in the formatting details of display output and command input. To a large extent, this depends on the computer used and on the available software. This is consistent with the fact that if the problem is too detailed, it might limit the scope for design creativity.

## B. Overview of ATC systems

ATC systems provide particularly pertinent examples of the complexity of hard real-time systems for several reasons. First, they are absolutely an important, widely known, and difficult application, and there are many of them throughout the world. Secondly, they contain instances of all distinguishing features of real-time systems. Finally, parts of them can be abstracted to illustrate various technical ideas. Figure 1 shows the basic control points in a typical air traffic control environment.

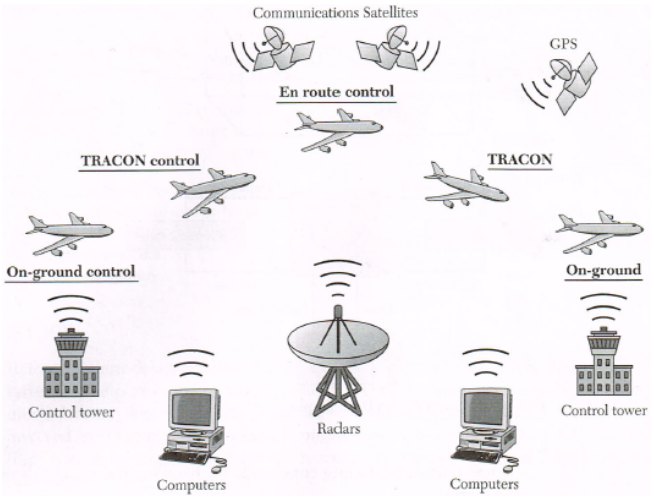


Figure 1. Typical air traffic control environment.

The airspace is divided into volumes called sectors. If an aircraft is flying in a controlled airspace, there is an air traffic controller (i.e., a person) responsible for that aircraft. As the aircraft moves from one sector to another, control of that aircraft passes from one controller to another. The sectors near airports are called Terminal Radar Approach Control (**TRACON**) sectors. In the figure, the air traffic controller directs traffic to the tower for landing, and away from the tower to **En Route** sectors after takeoff.
The major concerns of these systems are safety, efficiency, and performance. They are designed for preventing collisions and the occurrence of other hazards. For this purpose, air traffic controllers need to guarantee that adequate aircraft separation distances are maintained; and that weather hazards, natural obstacles, and restricted airspaces are avoided. Efficiency and performance concerns include maximizing airspace capacity and airport throughput, minimizing aircraft delays, and minimizing fuel consumption. At each individual center, there may be a variety of computer workstations and servers. There are also many forms of communications among components and between centers. Proposed changes in ATC include the ability for controllers to send flight plans directly from ground ATC to flight management systems on-board an aircraft. Radar will be augmented by aircraft that can report their position directly to ground control. Typically, the location is determined through communication with global positioning systems (GPS).

## C. Simplified version of an ATC

The aim of the simplified version of an ATC system is to keep track of all aircraft in 3D airspace and to guarantee that the aircraft always maintain a minimum separation distance. Figure 2 shows the computer system and its external environment.

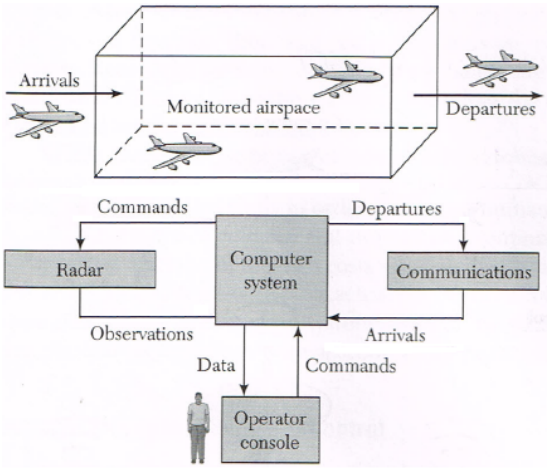


Figure 2. Simplified ATC system.

A radar tracks the position of each aircraft in the space by analyzing signals that bounce off each aircraft. When an aircraft enters the space, its presence is reported through a digital communications subsystem. On leaving the space, the computer system broadcasts a statement of its departure to neighboring sites. The communications subsystem can also be used to communicate directly with an aircraft. Interfaced to a human operator, a console displays the track of each aircraft and responds to operator commands. The operator commands include ones to request the status and descriptive data for an aircraft and to transmit messages to aircraft (e.g., to change direction to avoid a collision). Moreover, the radar can scan any portion of the airspace for a given space coordinates. If it detects an object at the commanded position, it returns a **hit**. An aircraft is considered lost if the radar fails to produce a hit. in this case, the operator must be notified, and corrective action must be taken. The radar is also used to scan the space for unknown objects and to identify lost aircraft. To

handle the diverse devices and the environment, some realistic timing constraints of the radar are as follows:

- It must track each aircraft in the space at a rate of at least one observation every 15 seconds per aircraft.
- The current position and track of each aircraft must be updated and displayed at least once every three seconds.
- A reply must be provided to an operator command within two seconds.

The main objectives of these timing constraints are threefold: 1) To provide the operators with enough time to understand the state of the airspace; 2) To control the positions of the aircraft by sending messages to them; and 3) To respond quickly to unusual or emergency situations such as potential collisions or overloads. The informally defined global objectives are translated into the above more detailed and precise constraints. Note also the inherent parallelism in the environment monitored by the computer. Radar observations, new arrivals, and operator commands can arrive independently and simultaneously. Similarly, the output signals, radar, display, and departures, are logically concurrent.

## D. Possible software architecture of an ATC

Figure 3 illustrates part of a possible implementation of an ATC in terms of periodic (denoted by $p$) and sporadic (denoted by $s$) processes. The major data structure is a **Track File** which is shared by several processes. It stores information such as current position, time-stamped previous positions, velocity, and identification for each aircraft in the space. The arrows in the figure denote data flow or message passing.

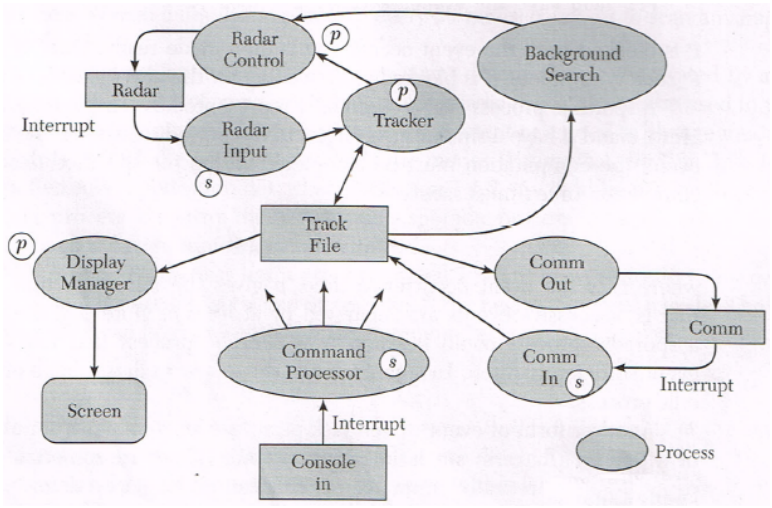


Figure 3. Partial design of an ATC software.

The main functions of each process are summarized as follows:

- Tracker: maintains the Track File, controls radar output, and processes radar hits.
- Background Search: searches airspace for unknown objects.
- Command Processor: receives and interprets the operator input and directs it to the appropriate internal process.
- Display Manager: Displays airspace contents, the response to operator commands, and any other output of interest to the operator.
- Radar Input and Output: controls and handles IO for the radar subsystem.
- Comm In and Out: controls and handles message IO for the communications subsystem.

Figure 4 illustrates a partial ATC specification with a data flow diagram (DFD). In this figure, some of the data and functional requirements for monitoring the entry, exit, and traversal of aircraft in an airspace are specified. The description is at a very high level, an "in-the-large" description. Many high-level elements and many lower-level details, however, are omitted.

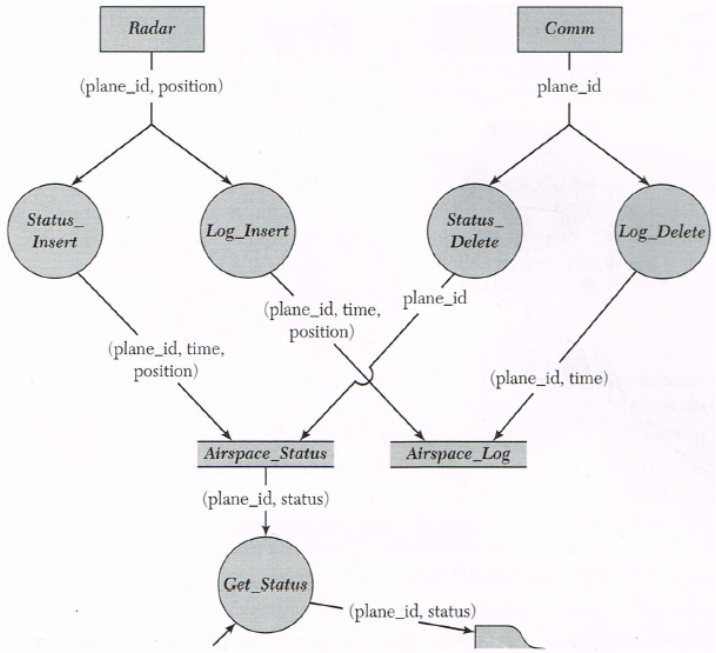


Figure 4. Partial ATC specification with a DFD.

Aircraft entering the space are detected by the ***Radar*** input. The ***Comm*** input identifies aircraft that leave the space. The current contents of the space are maintained in the data area ***Airspace_Status***. A log or history of the space is kept in the ***Airspace_Log*** storage. An ATC can request the display of the status of a pa1ticular plane through the ***Operator*** input. Note that the ***Get_Status*** function requires the operator input **plane_id** before it can retrieve the plane's **status** from ***Airspace_Status***.

## E. Project specification

As mentioned earlier, the purpose of the ATC is to guarantee the safety of aircraft traversing a given airspace (i.e., avoiding collisions) and to ensure their efficient traversal. To achieve these goals, the ATC must track or monitor each aircraft in the space and must be able to command an aircraft to change its path or its speed.

The safety constraint is defined in terms of the minimum separation. At any time, no two aircraft may be closer than a given distance from each other (e.g., 1000 feet in elevation and 3 miles in the horizontal plane). These numbers and units, and the others that are provided below, should be treated as environment and design parameters that could be changed easily. Note that in this project, we will not be too concerned with efficiency such us trying to maximize aircraft throughput. Nevertheless, the ATC should not degrade efficient operation except when the safety constraint may be otherwise violated. For the project, this means that unless two aircraft are projected to violate the minimum separation constraint, they will not be commanded to change their direction or speed.

### E.1. Environment

As shown in Figure 5, the airspace is a 3D rectangle which is assumed to be 100 × 100 miles in the horizontal plane and 25,000 feet high in elevation, bounded below by a horizontal plane at 15,000 feet, that is, above sea level. An aircraft enters the space flying in a horizontal plane at a constant velocity. It maintains its altitude and speed unless directed by the ATC to change. Thus, aircraft do not enter from above or below the space and fly in a horizontal plane under normal conditions. An aircraft entering the space is "handed- off" to the ATC by its controlling neighbor site. When an aircraft leaves the space, it is handed off to a neighboring ATC.

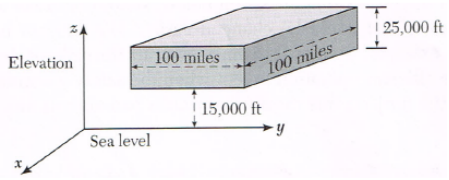


Figure 5. 3D airspace.

All aircraft have radio communication that can be tuned to the ATC, allowing pilots and ATC controllers to speak directly to one another. We will not be concerned with the details of this voice communications part of the system. Almost all aircraft are equipped with "transponders" that return identification and location coordinates to radars, and with digital communications that allow sending and receiving of electronic messages.

As illustrated in Figure 2, at the ATC site, there is a computer system with interfaces to a radar subsystem, digital communications subsystem, and display output and keyboard/mouse input for the human controller. Figure 6 shows the input and output interfaces provided by these subsystems.

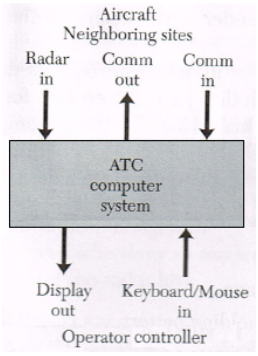


Figure 6. ATC computer input/output interfaces.

There is also a radio communications system that is independent of the computer system. Among other uses, this voice system provides a back-up to the computer system in case there is a problem with computer communications, radar tracking, or other failures, or in case an aircraft fails to correctly execute commands.

The radar subsystem scans the entire airspace every **15 seconds**, returning a list of "hits" in a buffer that can be accessed by the computer. As mentioned previously, hits are objects "seen" by the radar. The communications subsystem allows the ATC to send and receive digital messages.

The ATC controller which is a human operator, interacts with the system and aircraft by inputting commands through the keyboard and through voice on the radio. The system communicates with the controller through the output display. The display shows various views of the state of the airspace, echoes operator input, and lists electronic messages that are received.

### E.2. ATC computer inputs

Inputs to the ATC computer system can arrive from the radar subsystem, from the digital communications interface, and from the operator keyboard and mouse. The radar input is a hit list L with the following form and meaning:

$$L = size\ , Hit(1), Hit(2), \ldots, Hit(size).$$

Each hit element represents an object in the airspace. A hit $Hit(i)$ has the following structure:

$$Hit(i) = (aircraft\_id, (x, y\ , z))$$

where $aircraft\_id$ identifies the aircraft hit or seen by the radar, and $(x, y, z)$ gives the coordinates of the aircraft in the space. If an aircraft does not have a working transponder, the radar still returns a hit but with an $aircraft\_id$ field set to "unknown". The radar buffer is locked from computer access for a short time (e.g., 1 millisecond) near the end of the 15-second period while the hit data is copied to it.

Input messages through the communications system are either messages from individual aircraft in the space or hand-off messages from a neighboring space. In the first case, the input is either a response to an operator command or it may be a request for permission to change elevation or velocity (e.g., because of unstable air in the aircraft's current flight path). A hand-off message contains an aircraft identity and projected airspace entry data, such as time, location, and velocity at entry.

Messages are received in a queue that can be read and emptied by the computer system. Each message in the queue is of the form:

$$(sender\_id, message)$$

where $sender\_id$ is the name of the message sender and message is the contents of the message.

The operator input consists of commands or information requests directed either toward individual aircraft or to all objects in the space, of requests for data in the ATC system, of corrections or changes to data maintained by the system, and of hand-off messages to be sent to a neighboring ATC site. The operator can input any of the following commands to a designated aircraft:

- Change altitude by $\pm \boldsymbol{n} \times \boldsymbol{1000\ ft}$, where $\boldsymbol{n}$ is a positive integer.
- Increase or decrease its speed.
- Change direction in its horizontal plane.
- Enter or leave a **holding** pattern. A holding pattern is a closed flight path. One standard pattern is an oval or racetrack.
- Report its current position and velocity.

The operator can command all aircraft with either of the following:

- Enter or leave a holding pattern.
- Report aircraft identification, position, and velocity.

The operator requests to read or change internal systems data that represent the state of the airspace include the following:

- Add or delete an aircraft object.
- Change the position, elevation, or velocity of an aircraft.
- Display data record for a given aircraft.
- Project aircraft positions to $\boldsymbol{current\_time + n}$, where $\boldsymbol{n}$ is some integer number of seconds.

### E.3. ATC computer outputs

The system outputs consist of the operator display and the communications device. To send a message $m$ to a receiver $R$ over the communications subsystem, the computer system emits the following command:

$$send(R, m).$$

The receiver $R$ can be an aircraft or an ATC site. A message $m$ can be broadcast to any listening receivers with the following command:

$$broadcast(m).$$

The commands $send$ and $broadcast$ are both asynchronous (nonblocking). That is, the computer system proceeds immediately after executing the command. Hand-off messages are sent to neighboring sites. Directed and broadcast messages can be transmitted to the aircraft based on input commands from the operator. Messages are also generated internally based on the system's state. Particularly, a potential collision notification is sent if the system detects that aircraft are about to violate the minimum separation constraints. In this case, it is considered a back-up in case the operator fails to command the aircraft to take some safety action, or in case the aircraft fail to execute the operator commands.
In its normal mode, the display shows a plan (top) view of the airspace locating each aircraft in the space. Under operator control responding to command input, more complete data on each object are displayed. All electronic messages received from other aircraft and sites is also displayed. Moreover, if the separation constraint is violated, or will be violated within 3 minutes, a pictorial alarm such as flashing or red icons is output, along with pointers to the offending objects. Pictorial alarms are also displayed whenever an unidentified object is detected by the radar or an identified aircraft is lost (no radar hit).

### E.4. Software structure with timing constrains

The primary data structure in the ATC system is an airspace database containing a record of all aircraft in the space at the current time. Note that these data are only a model of the airspace, as known to the ATC computer system. They approximate the state of the actual space. How close this approximation is to reality depends on many factors, including the accuracy of the radar, communications reliability and transmission times, aircraft modelling assumptions, computer and IO overhead, and clock precision.
For each aircraft, there must be a structure including at least the following information:

$$aircraft\_id,\ current\_position,\ current\_velocity.$$

The software is responsible for performing the following functions:

- Display a plan view of the space every **five** (**5**) seconds, extrapolating if necessary to show the current position of each aircraft.
- Check all objects in the airspace for separation constraint violations at $\boldsymbol{current\_time + n}$ seconds, where $\boldsymbol{n}$ is an integer parameter. Display an alarm if a safety violation is found.
- Interpret and provide an initial response to all operator commands within **two** (**2**) seconds. For commands that require communications to and from aircraft, the initial response should occur after the appropriate message is sent to the communications subsystem.
- Receive and interpret input to the communications subsystem. This could be data sent from aircraft in the space or from neighboring sites.
- Maintain the airspace database. This requires functions to add, remove, and change the data of aircraft records.
- Store the airspace state in a history file every **sixty** (**60**) seconds. There should be enough information in this log to generate an approximation to the history of the airspace over time.
- Handle lost or unidentified objects, for example, by first broadcasting messages such as "Where are you?" or "Please identify yourself". Then, alerting the operator who can switch to the radio and attempt voice contact.
- Detect and handle failures including missed deadlines and failure of an aircraft to respond to an operator command.

### E.5. Implementation requirements

In addition to submitting their final team deliverables, student teams must meet the following implementation requirements:

- Fill in the missing details of the environment, inputs and outputs, and software functions and data.
- Describe the functional behavior of the software with data flow diagrams (DFDs).
- Specify the behavior of the ATC system and its environment, using either statecharts or communicating real-time state machines (CRSMs). The environment comprises any object that sends or receives data from the computer system. Your description should contain several concurrent machines, as well as several

**Commented [AT1]:** Skip this part in your report.

levels of hierarchy.

- Implement the ATC system and a simulator for the environment. Part of the simulator should permit entry of environmental data from the workstation terminal during execution. The active part of the system should consist of a set of periodic processes, where periodic polling is used to handle sporadic events. Assume that all processes or threads share single processor. Instead of accessing the computer system's clock for ATC software times and the environmental simulator's time, implement a virtual clock that can be set, started, stopped, and stepped from the terminal. It is suggested to use Real-time C++ which are object-class libraries specifically developed for C++. These libraries extend the standard C++ to provide an improved level of timing and control.
- Test your system under various operating conditions (low, medium, high, and over loads). The system load is determined primarily by the number of aircraft in the space, the degree of congestion in the space, and the amount of IO traffic.
- Measure the execution times of each process (best and worst cases if possible). Test for scheduling feasibility using rate monotonic fixed priority assignment.

## F. Final Team Deliverables

The deliverables consist of the following:

1) A copy of well-commented source codes and a set of sample runs.
2) Fifteen to twenty pages report specifying a high-level description of the ATC system. The report should also include a good summary of the results.
3) In your report, indicate the status of your programs by specifying whether:
   - The programs run with user-defined test cases.
   - The programs run with some errors.
   - The programs compile and run with no output.
   - The programs do not compile.

## Appendix

Execution times of programs can be obtained by direct measurements using software probes. This might be done to predict execution times, to check that run-time behaviors meet requirements, to validate an analytic method, to provide data for simulations, or to assess the actual performance of a system. Elements that may need to be timed include complete applications, individual programs, procedures and functions, code blocks, higher-level language constructs, and even single machine level instructions. In addition to performance data on applications software, execution times for various systems components are also typically desired or required. Examples of such systems code blocks or functions include interrupt handlers, context-switching mechanisms, schedulers, synchronization routines, clock and delay functions, and IO programs.

Suppose that $S$ is a self-contained body of code, containing no IO or other system calls. To obtain the execution time for $S$, we need a timer or clock. Let $Clock$ be a function that returns the current value of time. Then the simple way to measure the performance of $S$ is to call $Clock$ immediately before and after running $S$, as illustrated in the following test program:

```
-- Test Program 1
t_start := Clock;
S;
t_finish := Clock;
test_time := t_finish - t_start;
```

This might produce a satisfactory approximation to the desired value in the variable $test\_time$. However, one problem associated with the use of this simple technique is that the overhead in computing $test\_time$ is not considered. This overhead involves at least the time of two calls to the $Clock$ function and the assignments to $t\_start$ and $t\_finish$. The overhead time can be found by running the following control program, which essentially substitutes the null program for $S$ in the above:

```
-- Control Program
t_start := Clock;
-- This line is empty. It is the null program.
t_finish := Clock;
control_time := t_finish – t_start;
```

Thus, the running time of **S** can be obtained as follows:

```
S_time = test_time - control_time.
```

Some other sources of inaccuracy are due to interferences from the underlying operating system. Disabling interrupts during a timing run will ensure that no operating systems tasks preempt the test or control tasks. However, it is not always possible to disable interrupts. For instance, the clock itself is often driven by periodic "tick" interrupts. Finally, there may be significant errors due to the tick granularity and accuracy of the clock. Let $ct$ be the time returned from the $Clock$ function, $p$ be the tick period or tick interval, and $rt$ be the perfect real time. Assume that $ct$ is updated to real time at each tick and is immediately available at that time (no overhead on the update), as shown in Figure 7.

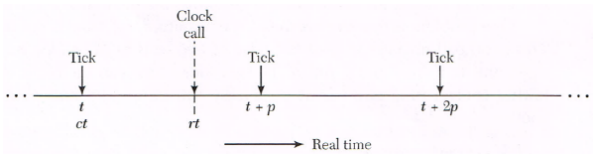


Figure 7. Relationships among clock variables.

If $Clock$ is called at the real-time $rt$, then the time $ct$ returned will satisfy the following relationship:

$$ct = rt - \delta p, \;\; 0 \leq \delta < 1.$$

Consequently, $\boldsymbol{ct}$ will be in error and early by as much as $p$. It is possible to compensate for this error by adding $0.5p$ to $ct$.
Consider now the difference $\Delta ct$ obtained from two calls of $Clock$:

$$\Delta ct = ct_2 - ct_1 = rt_2 - \delta_2 p - rt_1 + \delta_1 p = \Delta rt \pm \varepsilon$$

where $\varepsilon < p$. Thus, the difference between two clock times will have an error bounded by $p$. The $S\_time$ will then have a maximum error of $2p$, since it is the difference between two differences. In addition, the clock's tick may not occur exactly "on-time."
To overcome these problems, it is common to time the execution of many instances of $S$ in a loop, and subsequently divide by the loop count to get a better approximation to $S$'s time. A test program that executes $n$ instances of $S$ is shown below:

```
-- Test Program 2
t_start := Clock;
for i in 1 . . n loop
S;
end loop;
t_finish := Clock;
test_time := t_finish – t_start;
```

The control program is the same as above, substituting the $null$ statement for $S$ in the loop. The control program now measures the $Clock$ and loop overheads. We have the following relationship:

$$test\_time - control\_time = n \times S\_time + error$$

where $error$ is bounded by $2p$. Rearranging terms, this gives:

$$S\_time = [(\, test\_time - control\_time) + error]/n.$$

If the loop count $n$ is sufficiently large, the error term can be ignored.
For measuring higher-level language elements, it is also necessary to prevent the compiler from performing optimizations that might invalidate the timing data. Typical optimizations remove code from a loop or eliminate loops entirely if possible, for example, in the case of a $null$ statement in the loop. A typical solution to this problem is to embed $S$ and the $null$ program in procedures, denoted by $S\_proc$ and $null\_proc$, respectively, substitute a $while$ loop for the $for$ loop, and use a procedure to increment the loop index. For example, the test program may now become as follows:

```
-- Test Program 3
t_start := Clock;
i := 0;
while i < n loop
     S_proc;
     Increment (i);
end loop;
t_finish := Clock;
test_time := t_finish - t_start;
```

The presented techniques apply to a single execution encapsulated in $S$, and without explicit inputs or outputs. The inputs are defined implicitly by initializing variables of $S$ with input values. To obtain best or worst-case executions of a program, it would be necessary to select appropriate initializations (inputs) for these situations. While such a task is theoretically unsolvable in general, it can be approached in many practical instances. For example, standard software testing methods that attempt to provide control flow edge or path coverage may be employed.