MỤC LỤC

DANH MỤC BẢNG BIỂU

DANH MỤC HÌNH VẼ

LỜI MỞ ĐẦU

Trong những năm gần đây hoạt động phát triển giáo dục liên ngành đã trở nên cấp thiết hơn bao giờ hết, trong đó nổi bật là phương pháp giáo dục STEM (Science, Technology, Engineering và Math) chú trọng yếu tố thực hành và trải nghiệm. STEM giúp cho người học đặc biệt là các bạn học sinh bước đầu có kiến thức về khoa học công nghệ đồng thời phát triển khả năng tư duy logic, quan sát và sáng tạo, có cơ hội phát triển toàn diện mà không có cảm giác nặng nề khi học tập. Ngoài ra, STEM sẽ giúp các em có cái nhìn đa chiều hơn về cơ hội nghề nghiệp trong tương lai, mạnh dạn và chủ động hơn khi lựa chọn khối nghành ở cấp học cao hơn.

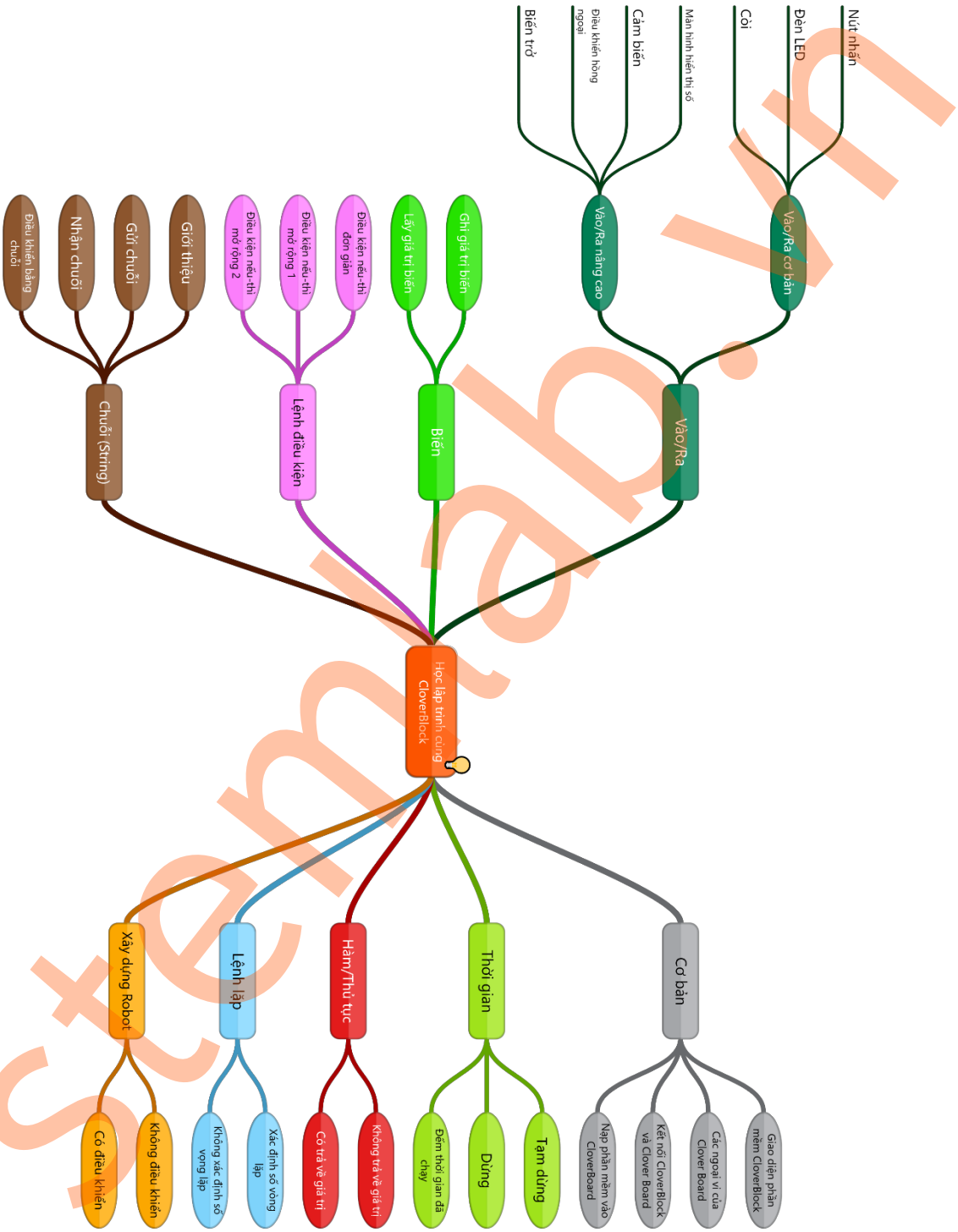
Clover là phần mềm lập trình dựa trên nền tảng mã nguồn mở Google Blockly do Clover Team phát triển để phục vụ học tập STEM. Phần mềm cho phép xây dựng chương trình thông qua thao tác kéo thả các lệnh đã được chuẩn hóa. Các bài toán tưởng chừng khó tiếp cận xung quanh cuộc sống có thể được triển khai một cách trực quan, thực hành và học tập dễ dàng.

Nội dung cuốn sách được thiết kế trên nền tảng phần mềm CloverBlock và phần cứng Clover Board do Clover Team nghiên cứu thiết kế. Mục tiêu cơ bản của cuốn sách là giúp học viên học lập trình thông qua những cách thức tiếp cận đơn giản và cơ bản để hình thành tư duy lập trình từ cơ bản đến nâng cao. Từ đó, giúp học viên có các kỹ năng về giải quyết vấn đề và kỹ năng lập trình cơ bản.

Nội dung cuốn sách bao gồm 11 nội dung chính sau:

- **Bài 1:** Các lệnh điều khiển vào ra cơ bản
- **Bài 2:** Các lệnh về thời gian
- **Bài 3:** Các lệnh điều khiển vào ra nâng cao
- **Bài 4:** Cấu trúc lệnh điều kiện (Nếu-Thì/If-Else)
- **Bài 5:** Biến (Variable)
- **Bài 6:** Thủ tục, hàm (Function)
- **Bài 7:** Lệnh lặp xác định
- **Bài 8:** Lệnh lặp không xác định
- **Bài 9:** Chuỗi văn bản
- **Bài 10:** Lập trình điều khiển Robot cơ bản
- **Bài 11:** Lập trình Robot có điều khiển

Cuốn sách đã được các thành viên của Clover Team đầu tư nghiên cứu, xây dựng nội dung rất chi tiết. Tuy nhiên, trong quá trình xây dựng và biên tập không thể tránh được những thiếu sót. Nhóm tác giả rất mong nhận được những phản hồi, ý kiến đóng góp của bạn đọc để nội dung cuốn sách được hoàn thiện hơn. Mọi ý kiến đóng góp xin gửi về ban quản trị Clover Team qua email: robolabcenter@gmail.com. Chúng tôi xin chân thành cảm ơn các thành viên của Clover Team đã đóng góp công sức xây dựng nội dung, xin chân thành cảm ơn các thầy cô của Khoa Kỹ thuật Điện tử I – Học viện Công nghệ Bưu chính Viễn thông đã cho chúng tôi những ý kiến đóng góp quý báu giúp chúng tôi hoàn thành được cuốn sách này.


Học lập trình cùng CloverBlock
Vào/Ra
Vào/Ra cơ bản
Nút nhấn
Đèn LED
Còi
Vào/Ra nâng cao
Cảm biến
Màn hình hiển thị số
Điều khiển hồng ngoại
Biến trở
Biến
Ghi giá trị biến
Lấy giá trị biến
Lệnh điều kiện
Điều kiện nếu-thì đơn giản
Điều kiện nếu-thì mở rộng 1
Điều kiện nếu-thì mở rộng 2
Chuỗi (String)
Giới thiệu
Gửi chuỗi
Nhận chuỗi
Điều khiển bằng chuỗi
Cơ bản
Giao diện phần mềm CloverBlock
Các ngoại vi của Clover Board
Kết nối CloverBlock và Clover Board
Nạp phần mềm vào CloverBoard
Thời gian
Tạm dừng
Dừng
Đếm thời gian đã chạy
Hàm/Thủ tục
Không trả về giá trị
Có trả về giá trị
Lệnh lặp
Xác định số vòng lặp
Không xác định số vòng lặp
Xây dựng Robot
Không điều khiển
Có điều khiển

CLOVERBLOCK LÀ GÌ?

Bạn muốn xây dựng một chương trình (phần mềm) để tạo ra một Robot hoặc một thiết bị điện tử hoạt động theo cách của riêng mình, để làm được việc đó thường bạn cần biết đến một ngôn ngữ lập trình như C/C++ dành cho Arduino hoặc một ngôn ngữ nào đó tương tự. Tuy nhiên, để học được một ngôn ngữ lập trình đủ để có thể làm được việc đó sẽ gây khó khăn cho các bạn đặc biệt với các bạn nhỏ và người không chuyên.

CloverBlock được Clover Team tạo ra để giúp người dùng vượt qua được rào cản đó và dễ dàng tiếp cận, tạo ra được các thiết bị và Robot hoạt động được theo ý đồ của riêng mình. Với CloverBlock bạn có thể viết một chương trình chỉ bằng cách kéo và thả các khối (Block). Hơn nữa, với CloverBlock các bạn hoàn toàn có thể thông qua chương trình được tạo ra bằng ngôn ngữ C/C++ tương ứng với các khối các bạn kéo thả để học được ngôn ngữ lập trình C/C++ nói chung và ngôn ngữ lập trình dành cho Arduino nói riêng.

CloverBlock được Clover Team phát triển trên nền tảng Blockly của Google và được cung cấp hoàn toàn miễn phí đến người dùng. Hiện nay, Clover Team đang tiếp tục nghiên cứu, phát triển để nâng cấp CloverBlock ngày càng trở nên hoàn thiện hơn.

CÀI ĐẶT CLOVERBLOCK

1. Bản cài đặt của CloverBlock có thể được tìm thấy và tải về tại trang:
   http://stemlab.vn/huong-dan-cai-dat-cloverblock/
   hoặc https://tinyurl.com/setupcloverblock
   (Hiện nay, CloverBlock mới hỗ trợ người dùng Window, người dùng Mac và Ubuntu sẽ sớm được Clover Team nghiên cứu và hỗ trợ)

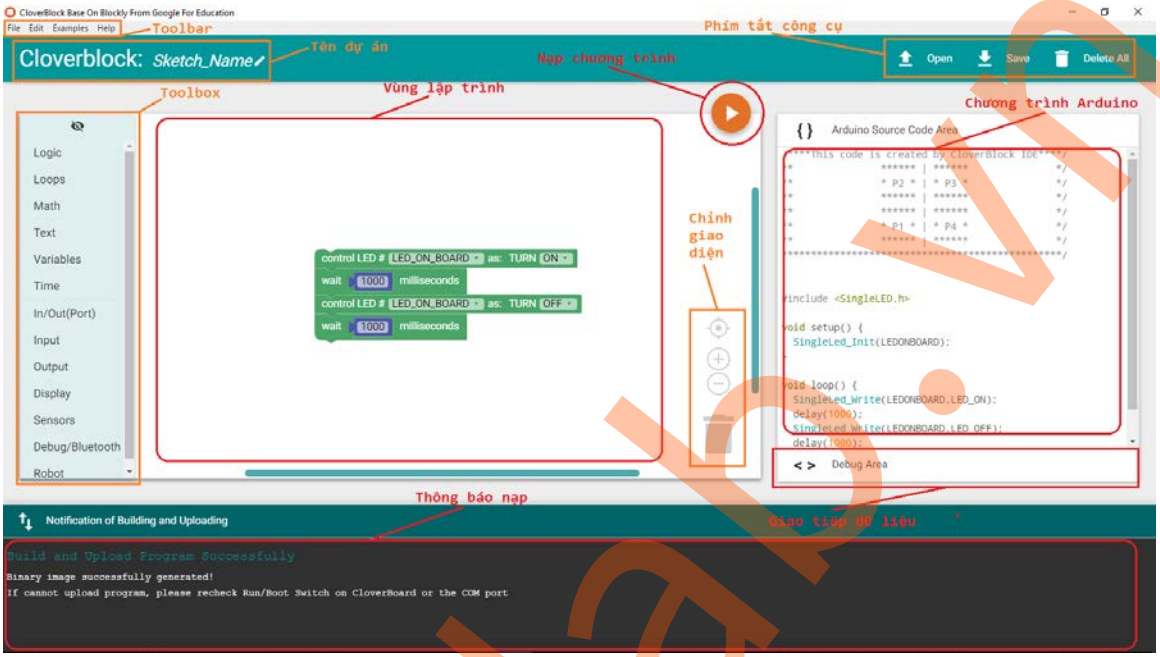
2. Sau khi tải về bản cài đặt người dùng nhấn đúp chuột vào file mới tải về và bắt đầu tiến hành cài đặt.

3. Trong quá trình cài đặt ở bước cuối phần mềm có thể yêu cầu cài đặt driver để kết nối với Clover Board, để có thể cài đặt driver thành công có thể cần tạm dừng các chương trình diệt virus và cho phép phần mềm được cài đặt driver này.

4. Sau khi cài đặt thành công sẽ xuất hiện Shortcut ở Desktop của màn hình Window. Người dùng tiến hành chạy thử nghiệm để kiểm tra.

GIỚI THIỆU CÁC CÔNG CỤ CỦA CLOVERBLOCK

Phần mềm CloverBlock có giao diện chính như sau:



Hình 1: Giao diện phần mềm CloverBlock

Toolbar: Gồm các tính năng quản lý file, chỉnh sửa (Undo, Redo, Copy, Paste,…), cài đặt, ví dụ và thông tin về phần mềm.

Tên dự án: Tên của file chương trình được lưu trữ.

Phím tắt công cụ: Bao gồm ba công cụ thường được sử dụng: Open (Mở một chương trình), Save (Lưu chương trình hiện tại), Delete All (Xóa toàn bộ lệnh trong vùng lập trình).

ToolBox: Cung cấp các khối lệnh (Block) để người dùng có thể lập trình. Các Block được chia thành các vùng theo chức năng của chúng.

Vùng lập trình: Chỉnh sửa, thêm bớt các Block để hoàn thành chương trình.

Chương trình Arduino: Hiển thị code Arduino bằng ngôn ngữ C/C++ do phần mềm sinh ra tương ứng với chương trình ở vùng lập trình.

Nạp chương trình: Tải chương trình ở vùng lập trình vào Board.

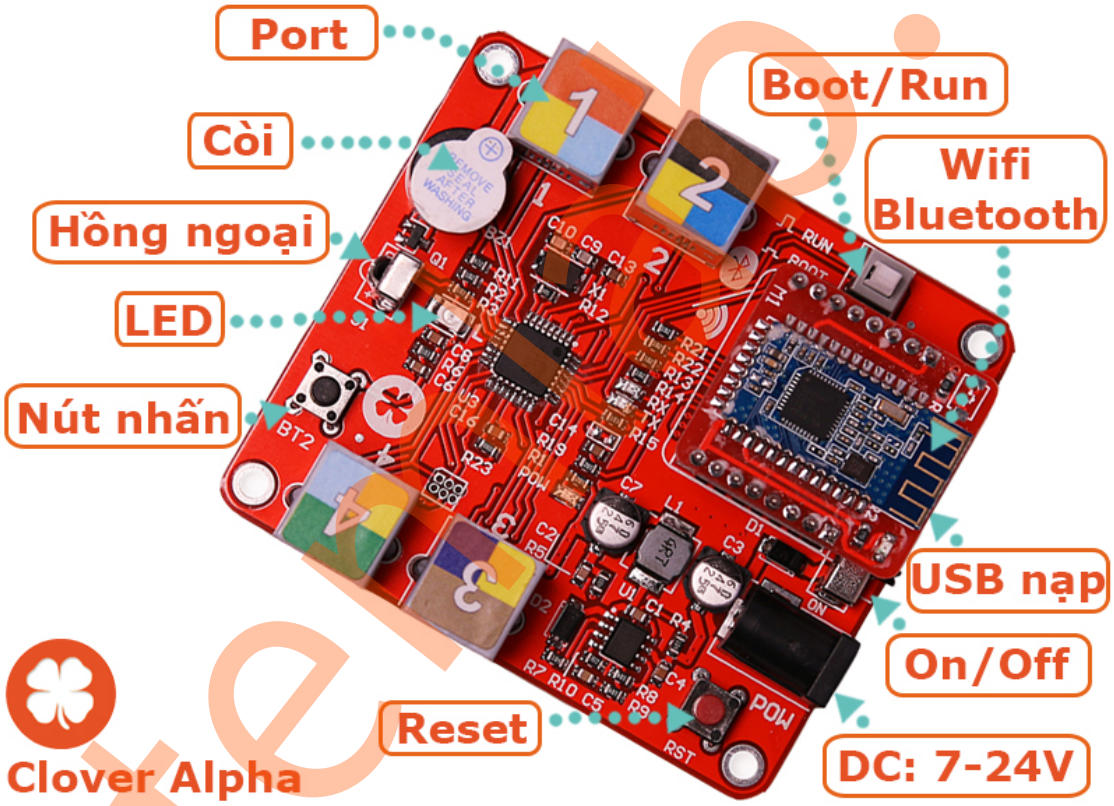
Chỉnh sửa giao diện: Phóng to, thu nhỏ giao diện lập trình, xóa các Block đang sử dụng.

Giao tiếp dữ liệu: Cho phép thực hiện truyền nhận dữ liệu giữa Board và máy tính.

Thông báo nạp: Hiển thị thông tin về quá trình nạp như thành công hoặc không thành công khi tiến hành nạp chương trình vào Board.

GIỚI THIỆU VỀ CLOVER BOARD

Clover Alpha Master Board là một trong những board chủ do Clover Team phát triển. Board hỗ trợ người dùng học, triển khai và thử nghiệm các ý tưởng lập trình của mình trên thiết bị thực tế. Board được tích hợp các thiết bị cơ bản như nút bấm, đèn LED, còi,… ngoài ra Board còn có các Port (Cổng) mở rộng dùng để kết nối với các thiết bị ngoại vi khác do Clover Team phát triển.



Hình 2: Cấu tạo của Clover Alpha Master Board

Clover Alpha Master Board được tích hợp một số thiết bị ngoại vi và cổng kết nối có tính năng cụ thể như sau:

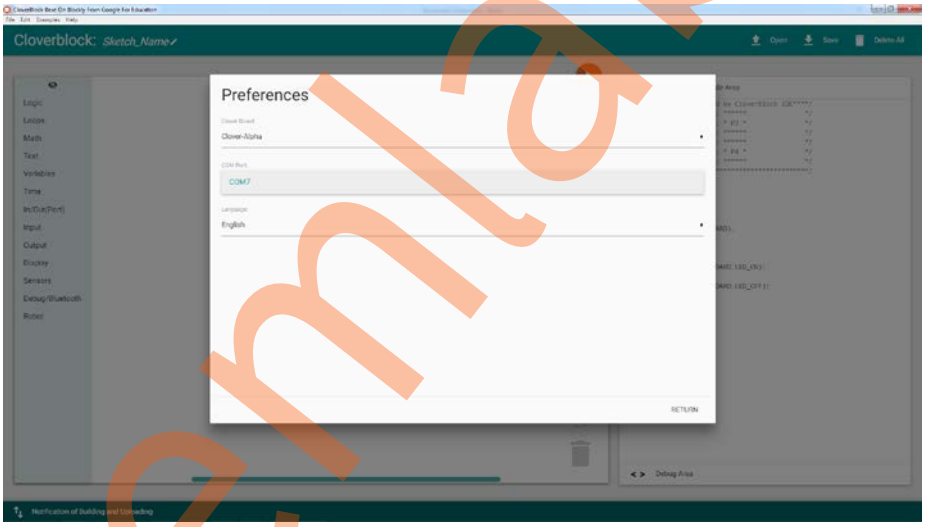
STT	Tên ngoại vi/cổng kết nối	Chức năng
1	**USB nạp**	Cung cấp nguồn 5V cho Board từ máy tính, sạc điện thoại, pin dự phòng… Nạp chương trình được lập trình từ CloverBlock và Board Giao tiếp truyền nhận dữ liệu giữa máy tính và Board
2	**Nút On/Off**	Cho phép cung cấp hoặc ngừng cung cấp nguồn cho Board khi sử dụng nguồn ngoài như: Pin, Adaptor… được kết nối qua Jack DC (3)
3	**DC Jack**	Cấp nguồn nuôi cho Board từ các nguồn ngoài máy tính như pin, adapter. Khuyến cáo sử dụng với nguồn từ 7-24V
4	**Nút Reset (Màu đỏ)**	Khởi động lại chương trình khi Board bị treo hoặc muốn chạy lại chương trình từ đầu.
5	**Nút Boot/Run**	**Boot**: Nạp chương trình từ máy tính xuống Board **Run**: Cho phép chạy chương trình khi có kết nối module Bluetooth hoặc Wifi Nếu không kết nối module Bluetooth/Wifi có thể để Board ở chế độ Boot mà vẫn có thể chạy được chương trình.
6	**Wifi/Bluetooth**	Cho phép Board mở rộng kết nối không dây sử dụng module Bluetooth hoặc Wifi nếu cần thiết
7	**Nút nhấn (Màu đen)**	Là một ngoại vi được sử dụng như dữ liệu đầu vào. Có nhiều chức năng và phụ thuộc vào người lập trình sử dụng.
8	**Đèn LED**	Được sử dụng như đầu ra điều khiển cơ bản: Bật, tắt hoặc điều chỉnh độ sáng.
9	**Còi**	Thiết bị ngoại vi âm thanh được sử dụng như đầu ra điều khiển.
10	**Hồng ngoại (IR)**	Thiết bị ngoại vi đầu vào, được sử dụng để Board giao tiếp với các loại điều khiển từ xa hồng ngoại như: Điều khiển điều hòa, điều khiển tivi,…
11	**Port (Cổng kết nối)**	Clover Alpha Board bao gồm 4 Ports (4 cổng kết nối). 4 cổng này được sử dụng để Board kết nối với các thiết bị ngoại vi (TBNV) khác như: TBNV nhập liệu đầu vào, TBNV đầu ra điều khiển, TBNV cảm biến, TBNV hiển thị, TBNV Robot… Các Port có thể kết nối với các thiết bị ngoại vi có màu kết nối tương đương.

Bảng 1: Danh sách ngoại vi của Clover Board

KẾT NỐI CLOVERBLOCK VỚI CLOVER BOARD

Clover Board và CloverBlock là hai công cụ cho phép người dùng sử dụng, khám phá về lập trình và chế tạo các thiết bị công nghệ mang tính cá nhân. CloverBlock là công cụ cho phép tạo ra phần mềm bằng tư duy logic và các thao tác kéo thả, Clover Board là công cụ để thực thi các ý tưởng đó bằng hình ảnh và hiện tượng thực tế. CloverBoard cần được kết nối với CloverBlock theo các bước như sau:

1. Kết nối CloverBoard với máy tính đã được cài đặt phần mềm CloverBlock thành công.
2. Mở phần mềm CloverBlock *(Bỏ qua bước này nếu phần mềm đã được mở trước đó)*
3. Trên thanh công cụ (**Toolbar**) chọn **Edit/Preferences**
4. Lựa chọn các thông tin về Board và kết nối:
   - Tại mục Clover Board chọn: **Clover-Alpha**
   - Tại mục COM Port chọn cổng COM của Clover Board. Nếu trong danh sách này không có cổm COM nào đề nghị kiểm tra lại kết nối, hoặc kiểm tra quá trình cài đặt Driver trong phần hướng dẫn cài đặt phần mềm.
   - Sau đó chọn "**Return**" để kết thúc cài đặt.



Hình 3: Giao diện cài đặt của CloverBlock

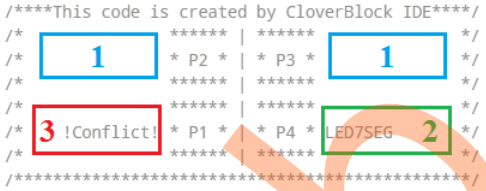
5. Kiểm tra trên Board đảm bảo công tắc "**Boot/Run**" đang ở chế độ "**Boot**" (Được nhấn)
6. Nhấn chọn "**Upload**" – "Nạp chương trình" và chờ cho đến khi không có biểu tượng xoay tròn trên nút nhấn này.
7. Bấm chọn khu vực "**Notification of Building and Uploading**" để xem nội dung thông tin nạp. Nếu xuất hiện "Build and Upload Program Successfully" thì quá trình kết nối giữa Board và phần mềm đã thành công. Nếu không thì quá trình kết nối chưa thành công, đề nghị người sử dụng xem lại các bước trên hoặc liên hệ với Clover Team để được hỗ trợ.

Website: http://stemlab.vn/
Fanpage Facebook: https://www.facebook.com/CloverSTEMLab/

NẠP CHƯƠNG TRÌNH TỪ CLOVERBLOCK VÀO CLOVERBOARD

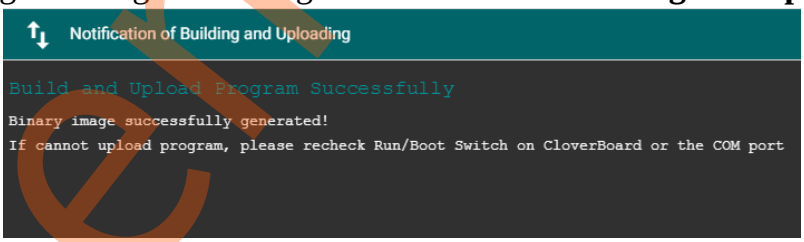
CloverBlock là phần mềm dùng để tạo chương trình điều khiển cho CloverBoard. Vì vậy, để thử nghiệm quá trình chạy của chương trình ta cần tiến hành nạp chương trình đã được xây dựng từ CloverBlock vào CloverBoard. Quá trình nạp chương trình gồm các bước sau:

1. Xây dựng hoặc mở chương trình cần nạp tại CloverBlock
2. Kết nối Cloverblock với Clover Board (Bỏ qua nếu đã thực hiện)
3. Thực hiện kết nối hoặc kiểm tra lại kết nối của các ngoại vi với Clover Board theo hướng dẫn trên phần mềm CloverBlock.



1: Các Port kết nối để trống, không sử dụng ngoại vi.
2: Kết nối ngoại vi có tên tương ứng vào Port này.
3: Có xung đột nhiều hơn 1 ngoại vi đang được khai báo sử dụng Port này. Cần xem lại chương trình.
Tiến hành sửa chương trình đến khi không còn xuất hiện "**Conflict**" và thực hiện kết nối ngoại vi theo hướng dẫn.

4. Chuyển trạng thái của công tắc "**Boot/Run**" về trạng thái **Boot**.
5. Nhấn nút nạp trên giao diện phần mềm CloverBlock và chờ đến khi biểu tượng xoay trên nút này dừng lại.
6. Kiểm tra thông tin thông báo ở vùng "**Notification of Building and Uploading**"



- Nếu có thông báo: Build and Upload Program Successfully, chương trình đã được nạp thành công. Tiến thực hiện bước 7.
- Nếu không có thông báo trên thì chương trình nạp chưa thành công. Có thể do lỗi về tư duy lập trình hoặc do kết nối chưa đảm bảo. Dựa vào hướng dẫn trên ô thông báo để tiếp tục giải quyết và thử lại các bước trên.

7. Chuyển trạng thái công tắc **Boot/Run** về trạng thái **Run** nếu chương trình sử dụng kết nối Bluetooth hoặc trao đổi dữ liệu với máy tính.
8. Thử nghiệm chương trình trên Board.

BÀI 1: CÁC LỆNH ĐIỀU KHIỂN VÀO RA CƠ BẢN

CloverBlock là phần mềm cho phép chúng ta lập trình bằng việc kéo thả sử dụng hình ảnh trực quan, ngoài ra với việc kết hợp với sử dụng CloverHardware còn cho phép người lập trình thấy được các lệnh thực thi trực quan bằng hình ảnh và hiện tượng xảy ra. Bài học này giới thiệu với bạn đọc về các thao tác lập trình để thực hiện điều khiển vào ra cơ bản sử dụng CloverBlock kết hợp với Clover Board.

1. Điều khiển trạng thái đèn/Trạng thái còi

Clover Board được tích hợp 2 thiết bị ngoại vi điều khiển đầu ra cơ bản là đèn LED và còi. Đây là 2 tín hiệu đầu ra điển hình thể hiện cho 2 trường hợp điều khiển tín hiệu đầu ra dưới dạng hình ảnh và âm thanh. Đây là hai dạng tín hiệu đầu ra thường gặp trong các thiết bị điện tử trong nhà. Ví dụ điển hình như: Đèn LED thể hiện trạng thái thiết bị đã được bật như Tivi, Máy giặt,… Còi thể hiện cảnh báo thao tác nhấn hoặc quá trình thực thi đã hoàn thành trên máy giặt.

a. Điều khiển trạng thái đèn

Lệnh điều khiển trạng thái đèn được đặt trong cụm Output của CloverBlock. Các Block có liên quan đến điều khiển trạng thái đèn được liệt kê trong bảng sau:

STT	Block	Vị trí	Chức năng
1		Output	Điều khiển trạng thái đèn LED thành bật hoặc tắt với trạng thái cần thay đổi là một Block (Biến hoặc giá trị cố định).
2		Output	Trạng thái của đèn dùng để cung cấp cho hàm điều khiển.

Bảng 1.1: Tập lệnh điều khiển trạng thái đèn LED

Để điều khiển trạng thái của đèn LED trên Board chúng ta thử nghiệm một đoạn chương trình như sau:

Hình 1.1: Chương trình điều khiển đèn LED sáng

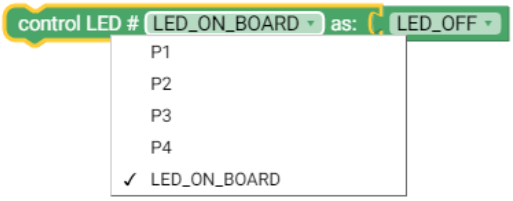
Sau đó, thực hiện nạp chương trình vào Board và quan sát trạng thái của đèn LED trên Board. (Tham khảo hướng dẫn nạp ở phần Kết nối CloverBlock với Clover Board)

Sau khi thử nghiệm xong, chúng ta thay đổi trạng thái điều khiển từ ON thành OFF như hình dưới và tiếp tục thử nghiệm nạp chương trình vào Board, quan sát trạng thái của đèn LED trên Board:

Hình 1.2: Chương trình điều khiển đèn LED tắt

Với 2 thử nghiệm trên chúng ta đã tiến hành điều khiển đèn LED về trạng thái bật hoặc tắt theo ý muốn. Ngoài ra, chúng ta có thể tiến hành thay đổi đèn LED cần điều

khiển bằng cách thử điều khiển đèn LED được kết nối với Board qua cổng kết nối (Port) bằng cách thay thế LED cần điều khiển như sau:

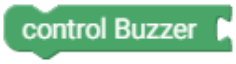


Hình 1.3: Lựa chọn đèn LED điều khiển

Các lựa chọn bao gồm: P1,P2,P3, P4 tương ứng với các cổng trên Board sau đó kết nối module đèn LED vào cổng tương ứng rồi thử nạp chương trình vào Board và quan sát hiện tượng.

b. Điều khiển trạng thái còi

Tương tự như điều khiển trạng thái đèn, các khối liên quan đến điều khiển trạng thái còi như sau:

STT	Block	Vị trí	Chức năng
1		Output	Điều khiển trạng thái của còi thành Bật (Phát âm) hoặc Tắt (Không phát âm)
2		Output	Trạng thái cần điều khiển còi được cung cấp cho Block điều khiển trạng thái còi. Block bao gồm 2 lựa chọn: BUZZER_ON và BUZZER_OFF

Bảng 1.2: Các lệnh điều khiển trạng thái còi

Để thực hiện điều khiển còi kêu chúng ta tiến hành thử nghiệm một chương trình đơn giản như sau:

Hình 1.4: Chương trình điều khiển bật âm trên còi

Thử nghiệm nạp chương trình vào Clover Board và kết quả xảy ra hiện tượng có tiếng kêu được phát ra từ còi nằm trên Board.

Để chuyển trạng thái còi về tắt (Không phát ra âm thanh) chúng ta thử nghiệm tương tự với đoạn chương trình sau:

Hình 1.5: Chương trình điều khiển tắt âm trên còi

Như vậy, sau lần thử nghiệm này chúng ta đã học được cách điều khiển phát sáng hoặc tắt đèn và phát hoặc tắt âm thanh bằng cách sử dụng CloverBlock và CloverBoard.

2. Đọc trạng thái của nút nhấn

CloverBoard được tích hợp 1 ngoại vi dữ liệu đầu vào cơ bản là nút nhấn. Nút nhấn là thiết bị thường gặp trong rất nhiều thiết bị dân dụng cũng như công nghiệp. Chúng ta có thể thấy các nút nhấn này ở nhiều thiết bị như: Tivi, Điều hòa, máy giặt,… Nút nhấn thường dùng để nhập tín hiệu điều khiển vào thiết bị để thiết bị hoạt động theo ý của người sử dụng.

Phần này giới thiệu một cách đơn giản để quan sát sự thay đổi của trạng thái nút nhấn (Đang nhả hay đang được nhấn) làm nền tảng thực hiện các bài toán sau này.

Các lệnh liên quan đến nút nhấn nằm trong cụm Input, bao gồm:

STT	Block	Vị trí	Chức năng
1		Input	Trả về True (Đúng) nếu nút nhấn ở trạng thái nhả (Không được nhấn), trả về False (Sai) nếu nút nhấn ở trạng thái nhấn
2		Input	Trả về True nếu mới có sự thay đổi từ Nhả thành Nhấn trên nút nhấn. Mỗi lần thay đổi trạng thái từ nhả về nhấn chỉ trả về True một lần.

Bảng 1.3: Các lệnh đọc trạng thái nút nhấn

Để thử nghiệm đọc dữ liệu đầu vào của tín hiệu nút nhấn chúng ta kết hợp sử dụng với các lệnh điều khiển tín hiệu đầu ra đã học ở phần trước và tạo thành một chương trình như sau:

Hình 1.6: Chương trình đọc trạng thái của nút nhấn (1)

Chương trình này được mô tả như sau: Đọc giá trị trạng thái của nút nhấn và dùng giá trị đó để điều khiển đèn LED. Đèn LED sẽ sáng khi nhận được đầu vào là True (Đúng - khi nút nhấn được nhả) và đèn sẽ tắt khi nhận được đầu vào là False (Sai – khi nút nhấn được nhấn). Tiến hành thử nghiệm nạp chương trình vào Board và thử nhấn nút, nhả nút để quan sát hiện tượng xảy ra trên đèn LED.

Với chương trình trên, khi chúng ta nhấn nút thì đèn sẽ sáng và khi chúng ta nhả nút thì đèn sẽ tắt, một số bạn sẽ muốn khi nào nhấn nút thì đèn tắt còn khi nào nhả nút thì đèn sẽ sáng. Vậy chương trình cần thay đổi như thế nào?

Cách đơn giản là chúng ta sử dụng một lệnh để đảo tín hiệu. Giống như chúng ta sử dụng từ **Không** (Not) trước các từ mô tả như trong ngôn ngữ chúng ta sử dụng.

Ví dụ: **Không sáng** nghĩa là **tối**, **Không tối** nghĩa là **sáng**, **Không đúng** nghĩa là **sai**, **Không sai** nghĩa là **đúng**,…

Lệnh đảo (NOT) nằm trong cụm Logic và có ký hiệu như sau:

Với lệnh **NOT** này, chúng ta có bảng đối chiếu như sau:

Tín hiệu	Đảo tín hiệu (NOT)
HIGH	LOW
LOW	HIGH
ON	OFF
OFF	ON
True	False
False	True

Bảng 1.4: Bảng trạng thái hoạt động của lệnh đảo (Not)

Từ đó, bài toán trên có thể được giải quyết bằng cách thêm một lệnh NOT vào trong chương trình đã có và chúng ta có kết quả như sau:

Hình 1.7: Chương trình đọc trạng thái của nút nhấn (2)

Chương trình trên có thể giải thích như sau: Đọc tín hiệu trạng thái từ nút nhấn, sau đó đảo tín hiệu của trạng thái nút nhấn (True thành False và False thành True) rồi đưa vào lệnh điều khiển trạng thái đèn LED.

Sau khi nạp chương trình vào Board và thử nghiệm kết quả cho thấy khi nút nhấn ở trạng thái nhả (không được nhấn), đèn LED ở trạng thái sáng và khi nút nhấn ở trạng thái được nhấn thì đèn LED ở trạng thái tắt như yêu cầu bài toán đã đặt ra ở trên.

Bài học này đã giúp chúng ta nắm được kiến thức cơ bản về đọc dữ liệu đầu vào từ nút nhấn ngoài ra còn giúp chúng ta khám phá về lệnh đảo (NOT).

3. Điều khiển độ sáng đèn

Một bài toán tiếp theo được đặt ra là thay đổi độ sáng của bóng đèn theo ý muốn. Ví dụ như khi ở trong bóng tối chúng ta cần bật đèn ở mức sáng cao nhưng khi ở trong phòng và đọc sách chúng ta chỉ cần ánh sáng vừa phải để tránh hại mắt,… Vậy làm thế nào để làm được điều này?

Ở bài toán này ngoài CloverBoard cần chuẩn bị thêm một module LED để có thể tiến hành thử nghiệm.

CloverBlock được tích hợp lệnh điều khiển độ sáng đèn LED, cụ thể về lệnh này như sau:

STT	Block	Vị trí	Chức năng
1		Output	Điều khiển ánh sáng của đèn, giá trị đầu vào từ 0-255.
2		Math	Cung cấp giá trị số được điền trong Block cho các Block khác

Bảng 1.5: Các lệnh liên quan đến điều khiển độ sáng đèn

Sử dụng 2 Block trên để tạo thành chương trình sau:

Hình 1.8: Chương trình mẫu điều khiển độ sáng đèn

Trước khi tiến hành thử nghiệm cần kết nói module đèn LED vào đúng cổng kết nối như hướng dẫn của phần mềm. Sau đó, nạp chương trình vào Clover Board và quan sát.

```
/****This code is created by CloverBlock IDE****/
/*               ****** | ******               */
/*               * P2 * | * P3 *               */
/*               ****** | ******               */
/*               ****** | ******               */
/*     SingleLED * P1 * | * P4 *               */
/*               ****** | ******               */
/*********************************************/
```

Hình 1.9: Hướng dẫn kết nối trên CloverBlock

Lần lượt thay đổi giá trị trong ô số (được khoanh đỏ) với các giá trị ngẫu nhiên trong khoảng từ 0-255. Các giá trị đề nghị thử nghiệm như sau: 0, 50, 100, 150, 200, 250… Khi các giá trị ở ô số được cung cấp khác nhau cho kết quả độ sáng được hiển thị trên đèn LED cũng khác nhau. Như vậy, chúng ta đã tiến hành điều khiển độ sáng đèn LED thành công.

Tóm lại, ở phần này chúng ta đã tìm hiểu được cách để thay đổi được mức sáng của đèn. Hiện tại, việc điều khiển giá trị độ sáng là cố định nhưng ở phần sau có thể kết hợp lệnh điều khiển độ sáng với các cảm biến để thực hiện thay đổi độ sáng đèn một cách tự động phù hợp với điều kiện ánh sáng của môi trường.

BÀI 2: CÁC LỆNH VỀ THỜI GIAN

Trước khi đi vào một số lệnh về vào ra nâng cao, ở bài này chúng ta cùng tìm hiểu các lệnh về thời gian trong CloverBlock. Các lệnh về thời gian là các hàm được sử dụng rất nhiều trong các chương trình.

Các lệnh về thời gian được nằm trong cụm Times của CloverBlock. Chi tiết về các lệnh trong CloverBlock như sau:

STT	Block	Vị trí	Chức năng
1		Times	Tạm dừng chương trình số milli giây (ms) tương ứng trước khi chạy lệnh tiếp theo.
2		Times	Tạm dừng chương trình số micro giây (µs) tương ứng trước khi chạy lệnh tiếp theo.
3		Times	Lấy về số milli giây (ms) kể từ khi Board được khởi động.
4		Times	Lấy về số micro giây (µs) kể từ khi Board được khởi động.
5		Times	Dừng chương trình tại đây cho đến khi Board được khởi động lại (Reset hoặc cấp lại nguồn).

Bảng 2.1: Các lệnh về thời gian của CloverBlock

Các ví dụ về đọc về thời gian kể từ khi Board được khởi động sẽ được trình bày ở một số bài toán sau. Bài học này tập trung vào 2 cụm lệnh tạm dừng chương trình và dừng chương trình.

1. Tạm dừng chương trình

Ở bài số 1 chúng ta đã tìm hiểu một số lệnh về điều khiển đầu ra và đọc dữ liệu đầu vào nhưng mới dừng lại ở điều khiển đầu ra ở một trạng thái cố định. Với các lệnh về thời gian chúng ta sẽ thử nghiệm thay đổi trạng thái đầu ra luôn phiên.

Ví dụ 1: Nhấp nháy đèn LED

Ý tưởng chính của bài toán này xuất hiện từ các bóng đèn trang trí mà chúng ta thường gặp trong thực tế. Chúng ta thấy các bóng đèn này thường không bao giờ được điều khiển ở một trạng thái sáng hoặc tắt cố định mà thường được nhấp nháy với mức độ nhanh chậm khác nhau để gây ra hiệu ứng bắt mắt hơn.

Để tiến hành bài toán này chúng ta thực hiện cho phép đèn sáng lên sau đó dừng lại một khoảng thời gian nào đó rồi tắt đèn đi sau đó lại dừng một khoảng thời gian và lặp lại từ đầu. Việc dừng lại một khoảng thời gian cho phép chúng ta quan sát được sự nhấp nháy đó. Chương trình được thiết kế cho bài toán này có dạng như sau:

Hình 2.1: Chương trình ví dụ tạm dừng chương trình

Tiến hành nạp chương trình vào Board và quan sát hiện tượng xảy ra trên LED. Kết quả cho thấy đèn LED thay đổi trạng thái giữa sáng cách nhau một khoảng thời gian là 1 giây.

Tiến trình của chương trình như sau: Lệnh đầu tiên thực hiện bật đèn LED lên. Sau đó, chương trình tạm dừng 1000ms (1 giây). Khi tạm dừng đủ 1000ms chương trình tiến hành chạy lệnh tắt đèn (Lệnh số 3) rồi tiếp tục lệnh tạm dừng 1000ms. Khi kết thúc 1000ms chờ này chương trình tự động quay lại lệnh đầu tiên (Lệnh bật đèn) và lặp lại chu trình cũ. Chu trình diễn ra lặp lại cho kết quả đèn LED liên tục được thay đổi giữa trạng thái sáng và tắt (Nhấp nháy)

Sau khi đã phân tích chúng ta có thể thay đổi thời gian của mỗi hàm tạm dừng thành các con số mong muốn và quan sát (Chú ý: Không nên sử dụng số âm). Chúng ta cũng có thể sử dụng hàm tạm dừng micro giây (µs) để thử nghiệm.

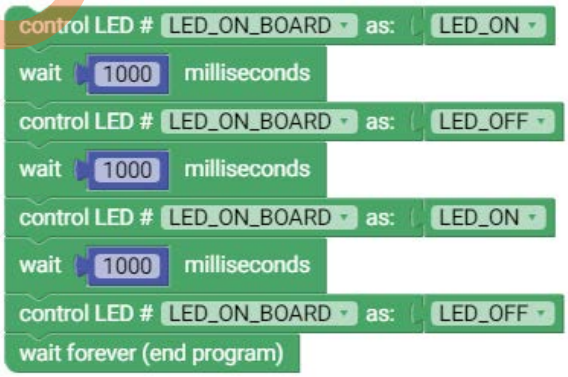
2. Dừng chương trình

Ở ví dụ điều khiển nhấp nháy đèn LED, đèn LED được nhấp nháy liên tục cho đến khi Board bị ngắt khỏi nguồn điện. Trong một số trường hợp chương trình chỉ yêu cầu thực hiện chạy một vài lệnh (Nhấp nháy đèn LED 1 vài lần) sau đó không chạy chương trình nữa. Vậy cần giải quyết như thế nào?

Câu trả lời là có thể sử dụng lệnh dừng mãi mãi để không cho phép chương trình tự động quay động lại lệnh đầu tiên của chương trình.

Ví dụ 2: Nhấp nháy đèn LED liên tục 2 lần rồi dừng lại không nhấp nháy nữa.

Bằng cách mở rộng từ chương trình ở ví dụ trên. Thực hiện điều khiển đèn LED nhấp nháy 2 lần sau đó sử dụng lệnh "**wait forever**" để chương trình dừng lại tại lệnh đó cho đến khi được ngắt nguồn hoặc được Reset. Chương trình cụ thể như sau:



Hình 2.2: Chương trình ví dụ về dừng chương trình

Nạp chương trình vào Board và quan sát hiện tượng. Kết quả cho thấy đèn LED chỉ nhấp nháy 2 lần (2 lần sáng và 2 lần tắt liên tiếp) sau đó dừng lại hẳn. Để quan sát lại hiện tượng này chúng ta có 2 cách: Nhấn nút Reset trên Board hoặc cấp nguồn lại cho Board (Ngắt kết nối Board với máy tính hoặc Gạt nguồn sang Off khi kết nối với Pin sau đó kết nối lại).

Ví dụ về lệnh "**wait forever**" sẽ được chỉ ra chi tiết hơn ở bài số 9 về lập trình điều khiển Robot.

BÀI 3: CÁC LỆNH ĐIỀU KHIỂN VÀO RA NÂNG CAO

Sau khi nắm vững các lệnh điều khiển vào ra cơ bản và các lệnh về thời gian, chúng ta tiếp tục tìm hiểu về các hàm vào ra cơ bản nâng cao hơn. Hiện nay, Clover hỗ trợ các thủ tục vào ra nâng cao như: Hiển thị số lên màn hình hiển thị số, Đọc dữ liệu đầu vào từ biến trở, Đọc dữ liệu đầu vào từ các cảm biến, Đọc dữ liệu đầu vào từ điều khiển hồng ngoại…

Bài học này chỉ giới thiệu về một số ngoại vi vào ra nâng cao, các ngoại vi còn lại sẽ được lần lượt giới thiệu ở các bài sau.

1. Điều khiển hiển thị số/ký tự lên màn hình số

Các lệnh về điều khiển hiển thị số lên màn hình số được mô tả chi tiết cụ thể như sau:

STT	Block	Vị trí	Chức năng
1		Display	Hiển thị một số nguyên hoặc một số thực lên màn hình hiển thị số tương ứng.
2		Display	Hiển thị thêm 1 ký tự đặc biệt (Một số chữ cái hoặc một số ký tự khác) vào một vị trí cụ thể trên màn hình hiển thị.

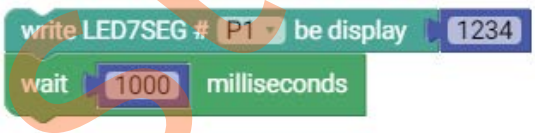
Bảng 3.1: Các lệnh điều khiển hiển thị màn hình số

Ở phần này, chúng ta sẽ cùng khám phá 2 ví dụ: Hiển thị một số và hiển thị một ký tự lên màn hình hiển thị số tương ứng.

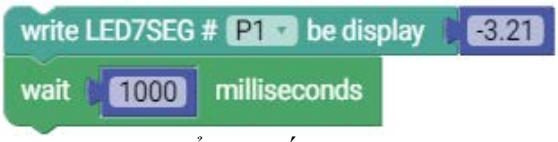
Ví dụ 1: Hiển thị số lên màn hình

Để thực hiện hiển thị một số nào đó lên màn hình chúng ta sử dụng lệnh số 1 ở bảng trên với đầu vào là một số tương ứng. Số đầu vào có thể là số nguyên hoặc số thực (Bao gồm cả số âm và số dương).

Theo như các ví dụ trên, ta thấy các lệnh sau khi được thực hiện sẽ tự động quay lại vị trí lệnh số 1 để thực hiện từ đầu. Vì vậy nếu chỉ có một lệnh thì lệnh đó sẽ được thực hiện một cách liên tục trong khoảng thời gian rất ngắn. Để các lệnh được gửi đến ngoại vi và có thời gian đáp ứng ta kết hợp sử dụng thêm hàm về thời gian để tạo độ trễ để các lệnh được lặp lại. Chương trình cụ thể như sau:



Hiển thị số nguyên dương



Hiển thị số thực âm

Hình 3.1: Chương trình hiển thị số lên màn hình hiển thị số

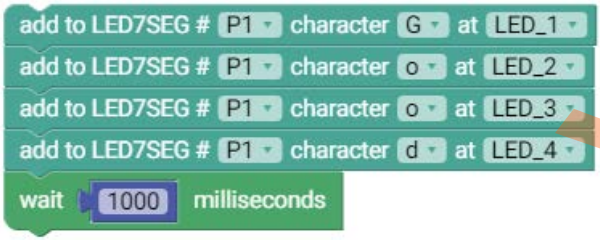
Trên đây ta sử dụng 2 ví dụ để hiển thị số nguyên và số thực. Sau mỗi lần lệnh hiển thị được điều khiển chương trình được tạm dừng 1000ms (1 giây) để số hiển thị lên màn hình và chúng ta có thời gian để quan sát trước khi lệnh hiển thị được lặp lại.

Thực hiện nạp chương trình vào Board và quan sát kết quả của 2 ví dụ trên để hiểu rõ hơn về lệnh này.

Ví dụ 2: Hiển thị ký tự lên màn hình

Ngoài hiển thị số, CloverBlock còn hỗ trợ chúng ta hiển thị một số ký tự, chữ cái đặc biệt lên màn hình hiển thị. Để có thể hiển thị được các ký tự ta sử dụng lệnh số 2 ở bảng trên. Ví dụ hiển thị chữ "Good" lên màn hình hiển thị số.

Chương trình cụ thể như sau:



Hình 3.2: Chương trình hiển thị ký tự lên màn hình hiển thị số

Ở ví dụ này, 4 ký tự G, o, o,d lần lượt được hiển thị lên các vị trí LED 1, 2, 3, 4 tương ứng trên màn hình hiển thị số. Tiến hành nạp chương trình vào Board để kiểm thử sau đó thay đổi nội dung các ký tự và tiếp tục thử nghiệm.

2. Đọc dữ liệu từ biến trở (Chiết áp)

Biến trở (Chiết áp) là một thiết bị phổ biến dùng để nhập dữ liệu đầu vào cho các thiết bị điện tử. Biến trở có thể dùng để: Điều chỉnh tốc độ quạt, điều chỉnh thời gian chạy lò vi sóng, điều chỉnh mức âm lượng... Tùy thuộc vào góc xoay tương ứng của người sử dụng mà thiết bị hoạt động với thời gian, tốc độ…. phù hợp.

Chi tiết về lệnh đọc giá trị biến trở cụ thể như sau:

STT	Block	Vị trí	Chức năng
1		Input	Đọc về giá trị tương ứng với vị trí của biến trở được kết nối ở Port tương ứng.

Bảng 3.2: Lệnh đọc dữ liệu từ biến trở (Chiết áp)

Ví dụ 3: Hiển thị giá trị đọc được từ biến trở lên màn hình hiểu thị số.

Để thực hiện ví dụ này ta sử dụng màn hình hiển thị số đã được học ở phần trước để hiển thị giá trị nội dung đọc được. Chương trình cho ví dụ này cụ thể như sau:

Hình 3.3: Chương trình đọc giá trị biến trở (Chiết áp)

Ví dụ trên được phân tích cụ thể như sau: Lệnh đọc giá trị biến trở ở cổng số 2 trả về một giá trị dạng số nguyên (Trong khoảng 0-1023). Giá trị này được sử dụng như đầu vào cho hàm hiển thị lên màn hình hiển thị số. Sau khi truyền giá trị này ra màn hình chương trình tạm dừng 100ms để có thể quan sát được giá trị đó. Khi kết thúc 100ms tạm dừng,

chương trình tự động lặp lại, đọc giá trị mới và tiếp tục hiển thị lên màn hình. Liên tục như vậy, khi biến trở được xoay giá trị tương ứng trên màn hình hiển thị cũng thay đổi theo.

Ví dụ 4: Điều khiển độ sáng đèn sử dụng biến trở.

Bài học trước đã giới thiệu lệnh điều khiển độ sáng đèn và ví dụ số 3 của bài học này đã giới thiệu chương trình đọc và hiển thị giá trị của biến trở lên màn hình hiển thị số. Vì thế, ta có thể tiến hành bài toán thay đổi độ sáng đèn bằng cách thay đổi góc xoay của biến trở.

Một vấn đề cần chú ý như sau: Giá trị để điều khiển độ sáng đèn LED cho phép nằm trong khoảng từ 0-255, trong khi đó giá trị của biến trở đọc được có giá trị từ 0-1023, hai giá trị này không hoàn toàn tương ứng với nhau. Vậy giải pháp là gì?

Ta có thể thấy: 1023:4 ≈ 255. Do đó, có thể sử dụng giá trị đầu vào của biến trở chia cho 4 trước khi đưa vào lệnh điều khiển độ sáng đèn LED.

Phép chia và các phép tính toán cơ bản nằm ở mục Math của CloverBlock. Một số lệnh về toán học cơ bản trong CloverBlock được giới thiệu ở bảng sau:

STT	Block	Vị trí	Chức năng
1		Math	Nhập một số nguyên hoặc một số thực tương ứng
2		Math	Thực hiện các phép toán có 2 phần tử: Cộng, trừ, nhân, chia, mũ
3		Math	Thực hiện các phép toán có 1 phần tử như: bình phương, trị tuyệt đối, lấy âm, logarit,...
4		Math	Thực hiện các hàm lượng giác như: sin, cos, tan, cotan,...
5		Math	Các giá trị toán học đã được quy định như: Số Pi, e,...

Bảng 3.3: Một số lệnh toán học cơ bản trên CloverBlock

Sau khi tìm hiểu về một số lệnh toán cơ bản, ta thiết kế được chương trình như sau:

Hình 3.4: Chương trình điều khiển độ sáng đèn sử dụng biến trở

Như ý tưởng trình bày ở trên, ta thực hiện đọc giá trị của biến trở sau đó trước khi đưa giá trị vào lệnh điều khiển độ sáng LED, ta thực hiện chia giá trị đọc được cho 4 để đảm bảo giá trị điều khiển nằm trong khoảng từ 0-255. Sau mỗi lần cập nhật chương trình được tạm dừng 100ms để các giá trị được thay đổi và người quan sát có thể đọc được giá trị số rõ ràng.

Như vậy, sau ví dụ trên ta đã nắm được cách sử dụng lệnh về đọc giá trị đầu vào từ biến trở để sử dụng điều khiển hoạt động của thiết bị phù hợp.

3. Đọc dữ liệu từ một số cảm biến

Cảm biến là các thiết bị ngoại vi cho phép thiết bị đọc và biết được các thông số của môi trường xung quanh từ đó có được những hiểu biết hoặc đưa ra các lệnh điều khiển tương ứng nếu cần thiết. Hiện nay, CloverBlock hỗ trợ một số loại cảm biến cơ bản, trong thời

gian sắp tới số lượng cảm biến sẽ tiếp tục được cập nhật để người dùng có thể sử dụng và thu thập được nhiều thông số hơn.

Dưới đây là chi tiết về các lệnh đọc dữ liệu cảm biến:

STT	Block	Vị trí	Chức năng
1		Sensors	Đọc giá trị khoảng cách từ vị trí đo đến vật cản thông qua cảm biến khoảng cách
2		Sensors	Đọc giá trị nhiệt độ của môi trường đo thông qua cảm biến nhiệt độ/độ ẩm
3		Sensors	Đọc giá trị độ ẩm của môi trường đo thông qua cảm biến nhiệt độ/độ ẩm
4		Sensors	Đọc giá trị ánh sáng thông qua cảm biến ánh sáng. Có 2 lựa chọn về chế độ: **VALUE**: Giá trị trả về trong khoảng 0-1023 **PERCENT**: Giá trị trả về trong khoảng 0-100%
5		Sensors	Đọc cảm biến ánh sáng về dưới dạng sáng/tối. Ngưỡng sáng/tối có thể điều chỉnh bằng biến trở trên cảm biến

Bảng 3.4: Các lệnh đọc cảm biến cơ bản

Các lệnh đọc giá trị cảm biến 1,2,3,4 ở bảng trên đều trả về giá trị dạng số nguyên nằm trong khoảng phù hợp với mỗi cảm biến. Ở phần này, các ví dụ tập trung vào sử dụng cảm biến nhiệt độ. Và các cảm biến khác cũng được sử dụng tương tự. Tương đương với ví dụ về đọc và hiển thị giá trị biến trở ta có chương trình đọc giá trị cảm biến nhiệt độ hiển thị lên màn hình hiển thị số như sau:

Hình 3.5: Chương trình đọc và hiển thị nhiệt độ

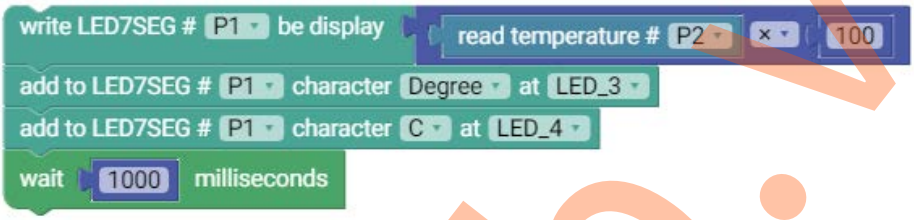
Thử nghiệm nạp chương trình vào Board với kết nối ngoại vi tương ứng và quan sát. Để thấy được sự thay đổi nhiệt độ chúng ta có thể sử dụng các thiết bị như: máy sấy, quạt, các thiết bị tạo nhiệt... đưa lại gần cảm biến nhiệt độ để quan sát sự thay đổi này.

Ngoài ra, ta có thể kết hợp hiển thị thông số nhiệt độ với các ký tự đặc biệt để thể hiện đơn vị của thông số giúp cho người quan sát có thể dễ dàng quan sát và hiểu được ý nghĩa của thông số đang được hiển thị lên màn hình.

Ý tưởng cụ thể như sau: Tiến hành điều khiển hiển thị nhiệt độ lên 2 vị trí số 1 và số 2 của màn hình hiển thị số sau đó thêm ký tự độ (°) và chữ cái C vào 2 vị trí số 3 và số 4 của

màn hình hiển thị. Tuy nhiên, theo kết quả của ví dụ trên, thông số nhiệt độ tự động được đẩy vào 2 vị trí số 3 và 4 của màn hình hiển thị. Vậy giải pháp của chúng ta như thế nào?

Ta thấy giá trị nhiệt độ môi trường là số có 1 hoặc 2 chữ số nên giá trị nhiệt độ được tự động đẩy về bên phải tức là chiếm 2 vị trí hiển thị 3 và 4. Một cách đơn giản để đưa 2 số này về vị trí 1 và 2 là ta thực hiện biến số đó thành số có 3 đến 4 chữ số bằng cách nhân giá trị nhiệt độ hiện tại với số 100. Ví dụ: 12x100=1200. Khi thực hiện phép nhân ta thấy thông số nhiệt độ đã được đưa sang vị trí số 1 và số 2, hai vị trí số 3 và số 4 được thay thế bởi 2 số 0. Sau đó, ta thay thế 2 ký tự không (0) bằng 2 ký tự (°C) tương ứng vào vị trí 3 và 4. Chương trình cụ thể như sau:



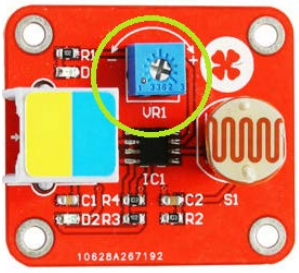
Hình 3.6: Chương trình đọc và hiển thị nhiệt độ kèm đơn vị

Như vậy, ta đã có thể thực hiện hiển thị được thông số nhiệt độ môi trường một cách rõ ràng lên màn hình hiển thị số. Với các cảm biến khác ta có thể sử dụng các lệnh tương ứng để thử nghiệm và khám phá.

Khác với các lệnh 1,2,3,4, lệnh số 5 không trả về giá trị số nguyên mà trả về giá trị dưới dạng lựa chọn: Sáng hoặc tối (Tương ứng với Đúng/Sai – True/False). Vì vậy, thay vì hiển thị giá trị này ta có thể đưa giá trị này vào để điều khiển trực tiếp trạng thái của đèn LED. Ví dụ cụ thể như sau:

Hình 3.7: Chương trình đọc cảm biến ánh sáng dạng số

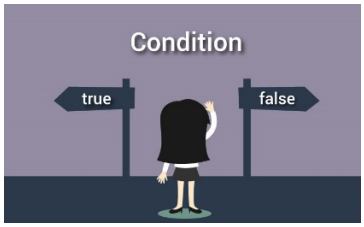
Tiến hành nạp chương trình vào Board, kết nối ngoại vi tương ứng sau đó thử nghiệm thay đổi độ sáng tác động vào cảm biến ánh sáng và quan sát sự thay đổi của trạng thái đèn LED tương ứng. Có thể thay đổi ngưỡng chuyển đổi giữa giá trị sáng – tối của cảm biến này bằng cách chỉnh biến trở xanh nằm trên module cảm biến ánh sáng.



Hình 3.8: Chỉnh ngưỡng cảm biến ánh sáng

Ví dụ này đã mô phỏng lại được hoạt động của hệ thống tự động bật đèn khi trời tối của hệ thống đèn đường. Chúng ta cũng có thể ứng dụng thiết kế này để thực hiện điều khiển tự động bật một số thiết bị theo điều kiện ánh sáng môi trường.

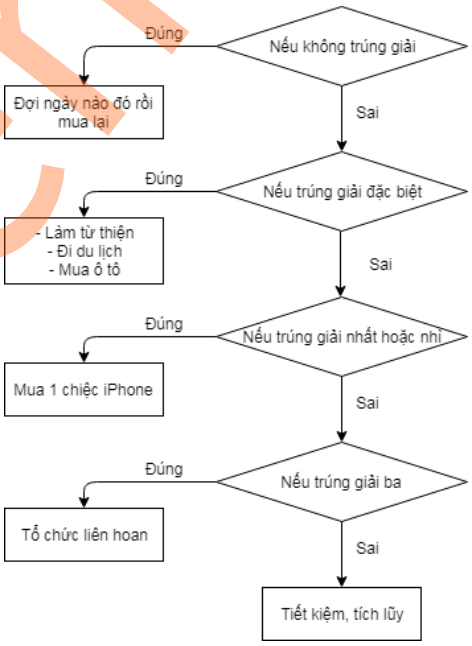
BÀI 4: CẤU TRÚC LỆNH ĐIỀU KIỆN (NẾU-THÌ / IF-ELSE)



Trong cuộc sống thực tế chúng ta gặp rất nhiều trường hợp về xử lý tình huống theo điều kiện (còn gọi là Nếu ... Thì ...). Ví dụ như: Nếu tôi giàu tôi sẽ đi du lịch vòng quanh thế giới. Hoặc đơn giản hơn giả sử bạn mua một vé xổ số bạn sẽ có cơ hội trúng giải đặc biệt, giải nhất, giải nhì... hoặc không trúng giải nào. Vậy bạn dự kiến sẽ làm gì nếu bạn trúng một trong các giải đó? Dưới đây là một ví dụ khi bạn quyết định mua một vé xổ số:

Nếu không trúng giải nào cả thì chúng ta sẽ mua giải vào một ngày nào đó khác để thử vận may. Ngược lại nếu trúng giải chúng ta có thể trúng 1 trong số rất nhiều giải. Nếu trúng giải đặc biệt (1 tỷ VND) thì tôi sẽ khuyên góp ủng hộ quỹ từ thiện 200 triệu đồng, đi du lịch xuyên Việt và mua một chiếc xe ô tô. Tất nhiên, tỷ lệ bạn trúng giải này không cao vì vậy bạn nên chuẩn bị kế hoạch dự phòng. Khi bạn không trúng giải đặc biệt bạn có thể trúng giải khác. Chúng ta tiếp tục giả sử. Nếu tôi chúng giải nhất hoặc giải nhì thì tôi sẽ mua cho mình 1 chiếc điện thoại iPhone. Nếu không trúng các giải trên mà chỉ trúng giải ba tôi sẽ mời gia đình một bữa liên hoan. Nếu không trúng các giải trên mà chỉ trúng các giải khác do số tiền quá nhỏ nên tôi để dành lại coi như khoản tích lũy.

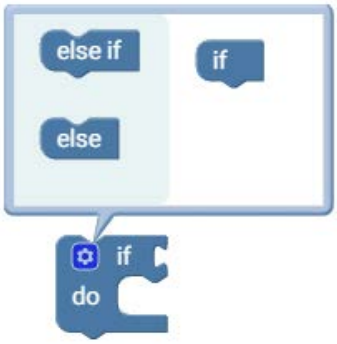
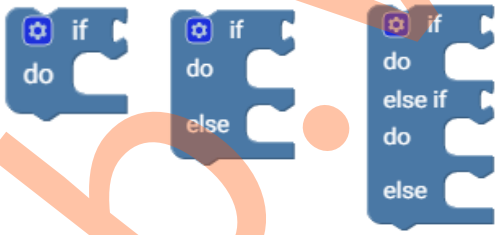
Với tư duy như trên chúng ta có thể vẽ được một sơ đồ tư duy như sau:



Hình 4.1: Ví dụ về một lưu đồ thuật toán

Với lưu đồ thuật toán trên ta có thể dễ dàng tra được cách xử lý trong tình huống xảy ra. Ngoài ví dụ trên, trong thực tế ta còn gặp rất nhiều ví dụ khác về các tình huống xử lý có điều kiện. Bài học này sẽ giới thiệu cách sử dụng các câu lệnh để xử lý các tình huống điều kiện trong lập trình với CloverBlock.

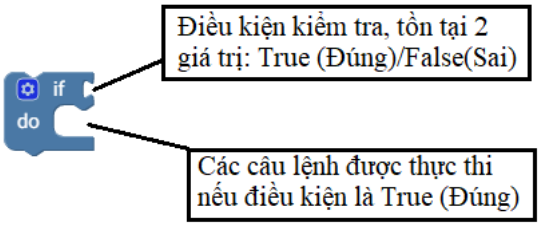
Các lệnh cơ bản liên quan đến điều khiển có điều kiện được mô tả ở bảng sau:

STT	Block	Vị trí	Chức năng
1		Logic	Lệnh điều khiển rẽ nhánh chương trình. Có thể kết hợp các thành phần để tạo thành nhiều dạng rẽ nhánh. Trong đó có 3 dạng chính:  *Dạng đơn giản* *Dạng mở rộng 1* *Dạng mở rộng 2*
2		Logic	Lệnh kiểm tra điều kiện đúng sai cơ bản. Bao gồm một số lệnh sau: So sánh bằng, so sánh hơn, so sánh hơn hoặc bằng, so sánh khác.
3		Logic	Lệnh kiểm tra kết hợp. Lệnh nào bao gồm 2 lựa chọn: And (Và), Or (Hoặc)
4		Logic	Lệnh đảo. Thực hiện biến True thành False và biến False thành True
5		Logic	Lệnh trả về 2 trường hợp điều kiện cơ bản là: True (Đúng) và False (Sai)

Bảng 4.1: Các lệnh điều kiện nếu-thì và lệnh liên quan

Chúng ta sẽ lần lượt tìm hiểu các lệnh điều kiện này.

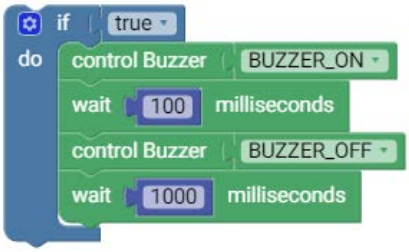
1. Lệnh điều kiện nếu-thì đơn giản



Hình 4.2: Cấu trúc lệnh điều kiện nếu-thì đơn giản

Hoạt động của lệnh điều khiển nếu-thì đơn giản được mô tả ở hình vẽ trên. Câu lệnh gồm 2 lệnh đầu vào: Điều kiện kiểm tra là lệnh trả về giá trị true hoặc false, các câu lệnh thực thi (Có thể gồm nhiều lệnh kết hợp). Khi điều kiện kiểm tra trả về là **true** (đúng) thì các câu lệnh thực thi bên trong lệnh điều kiện nếu-thì được thực hiện, ngược lại nếu điều kiện kiểm tra là **false** (sai) thì các câu lệnh đó không được thực hiện.

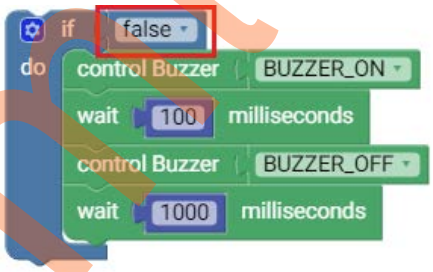
Tiến hành thử nghiệm với ví dụ đơn giản sau:



Hình 4.3: Chương trình điều kiện nếu-thì với điều kiện "true"

Tiến hành nạp chương trình trên vào Board và cùng kiểm tra hiện tượng. Ở chương trình trên, điều kiện kiểm tra là **true** (Đúng). Vì vậy, 4 lệnh bên trong lệnh điều kiện sẽ được thực hiện. Sau khi thực hiện xong lệnh tự động thoát ra khỏi lệnh điều khiển nếu-thì và vì không có lệnh nào phía dưới nên chương trình tự động quay lại lệnh điều kiện ban đầu và tiếp tục chạy. Chính vì vậy, trên Board liên tục phát ra tiếng kêu đan xen giữa bật và tắt được điều khiển bởi các lệnh nằm bên trong lệnh điều kiện nếu-thì.

Để xem xét rõ hơn về cách hoạt động của lệnh rẽ nhánh chúng ta thử thay đổi điều kiện **true** thành **false** và thử nghiệm lại:

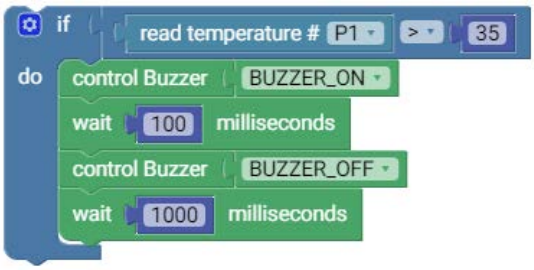


Hình 4.4: Chương trình điều kiện nếu-thì với điều kiện "false"

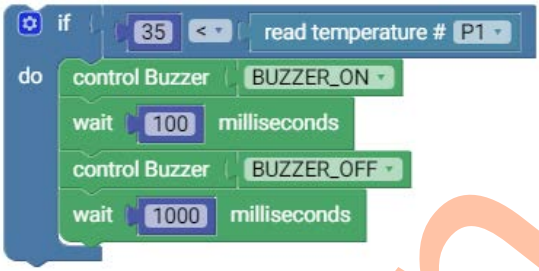
Ở đoạn chương trình này chúng ta đã thay đổi điều kiện kiểm tra từ true (Đúng) về thành false (Sai). Vì điều kiện kiểm tra là **false** nên các câu lệnh bên trong khối lệnh điều khiển nếu-thì không được thực thi, do đó không có âm thanh phát ra từ còi của Board.

Ví dụ 1: Từ ví dụ trên ta mở rộng bài toán thành như sau: Phát tín hiệu cảnh báo ra còi nếu nhiệt độ môi trường cao hơn 35°C.

Ở ví dụ trên ta thấy nếu điều kiện kiểm tra đúng thì có âm thanh phát ra từ còi, nếu sai thì sẽ không có âm thanh phát ra. Áp dụng vào ví dụ này, ta cần thực hiện câu lệnh điều kiện thỏa mãn nếu nhiệt độ đọc từ cảm biến lớn hơn 35°C thì kết quả kiểm tra điều kiện phải true (đúng) ngược lại nếu nhiệt độ nhỏ hơn hoặc bằng 35°C thì kết quả trả về phải là false (sai). Từ yêu cầu đó chúng ta có thể sử dụng lệnh so sánh và ở đây là so sánh hơn (>) hoặc (<) để thực hiện yêu cầu này. Chương trình như sau:



Chương trình sử dụng so sánh lớn hơn



Chương trình sử dụng so sánh nhỏ hơn

Hình 4.5: Chương trình cảnh báo nhiệt độ cao

Tiến hành thử nghiệm cả 2 chương trình trên board với cảm biến nhiệt độ/độ ẩm được kết nối ở cổng số 1và kiểm tra hoạt động của chương trình. Có thể thấy việc sử dụng > và < có thể coi là tương đương vì ta có thể thay đổi vị trí của 2 đối số được dùng trong phép so sánh.

Ví dụ trên chỉ sử dụng phép so sánh hơn nhưng khi gặp các bài toán khác ta có thể linh động sử dụng các phép so sánh khác. Chi tiết các phép so sánh được liệt kê ở bảng sau:

Phép so sánh	Ý nghĩa
=	Trả về True nếu 2 đối tượng so sánh bằng nhau
≠	Trả về True nếu 2 đối tượng so sánh không bằng nhau
<	Trả về True nếu đối tượng so sánh bên trái nhỏ hơn đối tượng so sánh bên phải
≤	Trả về True nếu đối tượng so sánh bên trái nhỏ hơn hoặc bằng đối tượng so sánh bên phải
>	Trả về True nếu đối tượng so sánh bên trái lớn hơn đối tượng so sánh bên phải
≥	Trả về True nếu đối tượng so sánh bên trái lớn hơn hoặc bằng đối tượng so sánh bên phải

Bảng 4.2: Các phép so sánh

Ví dụ 3: Giả sử chuẩn bị bước vào vụ lúa và người nông dân cần tiến hành ủ thóc để hạt thóc nảy mầm trước khi gieo cấy. Nhiệt độ ủ mầm lúa tối ưu được các chuyên gia nông nghiệp khuyến cáo nằm trong khoảng từ 28°-35°C. Trong thực tế, có thể do điều kiện môi trường mà điều kiện không đạt được như các chuyên gia khuyến cáo vì thế cần thiết phải có sự can thiệp để điều chỉnh nhiệt độ gieo ủ phù hợp. Bài toán của chúng ta là thiết kế thiết bị cảnh báo nếu điều kiện trên bị vi phạm (Không tốt cho sự phát triển của mầm lúa) thì phát ra âm thanh cảnh báo để người nông dân kịp thời can thiệp.

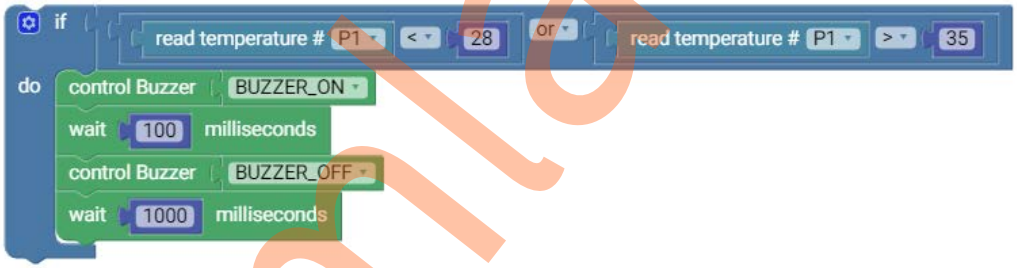
Thay vì điều kiện đơn lẻ chỉ cảnh báo khi nhiệt độ lớn hơn 35°C như ví dụ số 2. Ở đây điều kiện 2 phía nghĩa là khoảng an toàn là một khoảng giá trị bị chặn cả 2 đầu.

Trước khi tiến hành tìm cách giải cho bài toán trên ta xem xét 2 lệnh kiểm tra điều kiện kết hợp sau:

<table>
<tr><th>Phép kết hợp</th><th>Ý nghĩa</th></tr>
<tr><td>and (và)</td><td>Trả về True cả 2 đối tượng kết hợp có giá trị là true, chỉ cần xuất hiện ít nhất 1 đối tượng cho kết quả false thì lệnh này trả về giá trị false
<table>
<tr><th>Trường hợp</th><th>Kết quả “and”</th></tr>
<tr><td>True and True</td><td>True</td></tr>
<tr><td>True and False</td><td>False</td></tr>
<tr><td>False and True</td><td>False</td></tr>
<tr><td>False and False</td><td>False</td></tr>
</table></td></tr>
<tr><td>or (hoặc)</td><td>Trả về True khi ít nhất 1 trong 2 đối tượng có giá trị là true. Kết quả trả về là False khi cả 2 đối tượng kết hợp đều có giá trị là False.
<table>
<tr><th>Trường hợp</th><th>Kết quả “or”</th></tr>
<tr><td>True or True</td><td>True</td></tr>
<tr><td>True or False</td><td>True</td></tr>
<tr><td>False or True</td><td>True</td></tr>
<tr><td>False or False</td><td>False</td></tr>
</table></td></tr>
</table>

Bảng 4.3: Chi tiết về lệnh kết hợp and và or

Ta thấy điều kiện để phát ra cảnh báo trong trường hợp này: Nhiệt độ nhỏ hơn 28°C hoặc lớn hơn 35°C. Vì thế, ta sử dụng phép kết hợp **or** để tiến hành giải quyết bài toán này. Chương trình cho bài toán này như sau:



Hình 4.6: Chương trình cho thiết bị cảnh báo nhiệt độ ủ mầm lúa

Ở đây, chương trình tiến hành 2 phép so sánh: So sánh nhỏ hơn nhiệt độ với 28 và so sánh lớn hơn nhiệt độ với 35, sau đó chúng ta tiến hành **or**(hoặc) hai điều kiện này. Do đó, kết quả điều kiện là **true** khi một trong 2 điều kiện nhắc ở trên là true (Một trong hai điều kiện đúng). Trong điều kiện thực tế thì 2 điều kiện trên không thể cùng đúng (True) vì nhiệt độ không thể vừa lớn hơn 35 vừa nhỏ hơn 28 được. Sử dụng lệnh or đảm bảo khi có nhiệt độ nằm ngoài khoảng an toàn thì thiết bị sẽ đưa ra tín hiệu âm thanh cảnh báo.

Như vậy, với sự kết hợp điều kiện bằng lệnh **or** ta đã tạo được thiết bị cảnh báo nhiệt độ cho việc ủ mầm lúa của người nông dân. Ngoài ra, có thể áp dụng bài toán này với nhiều đối tượng thực tế khác với các điều kiện dữ liệu đầu vào khác.

Bài toán mở rộng: *Điều kiện tốt để bảo quản tủ thuốc là nhiệt độ nhỏ hơn 30°C và độ ẩm không quá 75%. Hãy thiết kế thiết bị cảnh báo bằng âm thanh khi điều kiện bảo quản vi phạm điều kiện trên.*

Ví dụ 4: Từ ví dụ trên ta tiếp tục phát triển thành một bài toán cảnh báo khác, đó là cảnh báo cháy rừng. Cháy rừng rất dễ xảy ra trong điều kiện nhiệt độ môi trường cao kèm theo độ ẩm trong không khí thấp tạo điều kiện cho các đám cháy dễ bùng phát. Giả sử hệ thống cảnh báo cháy rừng có điều kiện cảnh báo hoạt động khi đồng thời nhiệt độ cao hơn 40°C và độ ẩm không khí nhỏ hơn 50% *(Đây không phải là điều kiện thực tế và có thể phức tạp hơn)*. Với bài toán này chúng ta cần giải quyết như thế nào?

Cách 1: Sử dụng lệnh điều khiển điều kiện lồng.

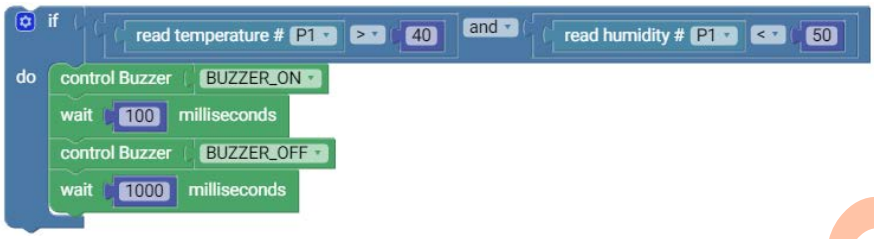
Cách đơn giản là lần lượt kiểm tra các điều kiện này trong các vòng lệnh điều kiện khác nhau. Vì điều kiện cảnh báo chỉ xảy ra khi cả 2 điều kiện xảy ra đó là: Nhiệt độ lớn hơn 40°C và Độ ẩm nhỏ hơn 50%, vậy nếu nhiệt độ nhỏ hơn hoặc bằng 40°C thì chắc chắn không đưa ra cảnh báo. Vậy nên không cần kiểm tra điều kiện về độ ẩm. Ngược lại nếu, nhiệt độ lớn hơn 40°C thì chúng ta tiến hành tiếp tục kiểm tra điều kiện về độ ẩm để quyết định đưa ra cảnh báo. Từ tư duy này, ta xây dựng chương trình như sau:

Hình 4.7: Chương trình cảnh báo cháy rừng một ngưỡng sử dụng điều kiện lồng

Chương trình sử dụng hai lệnh điều kiện nếu-thì và lệnh điều kiện thứ hai nằm trong tổ hợp lệnh thực thi của lệnh điều kiện thứ nhất, đây được gọi là điều kiện lồng. Có thể sử dụng lệnh điều khiển lồng với nhiều hơn 2 lệnh điều kiện trong trường hợp cần thiết. Ta thấy: Nếu nhiệt độ nhỏ hơn hoặc bằng 40°C thì cách lệnh thực thi bên trong câu lệnh nếu-thì thứ nhất không được thực thi vì vậy tổ hợp lệnh cảnh báo *(Màu xanh lá)* chắc chắn không được thực thi. Nếu nhiệt độ lớn hơn 40°C thì lệnh nếu-thì thứ hai được kiểm tra. Nếu độ ẩm lớn hơn hoặc bằng 50% nghĩa là điều kiện kiểm tra cũng không đúng, vì thế tổ hợp lệnh cảnh báo *(Màu xanh lá)* cũng không được thực thi. Ngược lại, nếu điều kiện độ ẩm thỏa mãn nhỏ hơn 50% thì lúc này tổ hợp lệnh cảnh báo được thực thi và chúng ta nghe được tín hiệu âm thanh cảnh báo. Như vậy, các lệnh cảnh báo chỉ được thực thi khi cả hai điều kiện nhiệt độ lớn hơn 40°C và độ ẩm nhỏ hơn 50% là đúng.

Cách 2: Sử dụng phép kết hợp **and**.

Ngoài cách sử dụng lệnh điều khiển lồng ta có một giải pháp khác là sử dụng phép kiểm tra điều kiện kết hợp **and** (và). Ta thấy điều kiện cảnh báo chỉ được bật khi đồng thời hai điều kiện *Nhiệt độ lớn hơn 40°C* và *Độ ẩm nhỏ hơn 50%* xảy ra. Vì thế, ta sử dụng phép kiểm tra điều kiện hợp **and** (kiểm tra đồng thời) và có chương trình như sau:



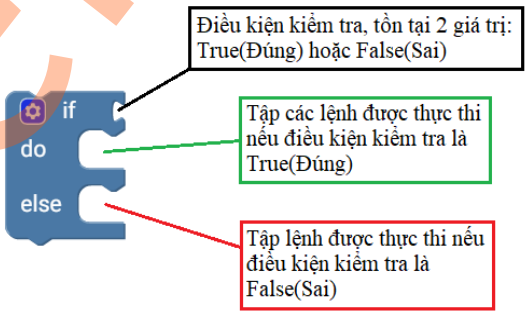
Hình 4.8: Chương trình cảnh báo cháy rừng một ngưỡng sử dụng kết hợp "and"

Chương trình được phân tích như sau: Khi nhiệt độ lớn hơn 40°C thì kết quả của lệnh kiểm tra thứ nhất trả về kết quả là **true**, ngược lại thì trả về **false**; Khi độ ẩm nhỏ hơn 50% thì lệnh kiểm tra thứ hai trả về kết quả là **true**, ngược lại thì trả về là **false**. Với phép kết hợp **and** kết quả trả về là **true** khi cả hai điều kiện: *Nhiệt độ lớn hơn 40°C* và *Độ ẩm nhỏ hơn 50%* xảy ra thì cảnh báo được bật. Nếu chỉ một trong hai điều kiện trên không đúng thì kết quả sẽ trả về là **false** tương ứng với lệnh cảnh báo được tắt. Điều kiện này phù hợp với điều kiện yêu cầu của bài toán. Để kiểm nghiệm ta thực hiện nạp chương trình vào Board và thử nghiệm.

Như vậy, sau một số ví dụ cụ thể ta đã tìm hiểu được về cách sử dụng lệnh điều khiển điều kiện nếu-thì đơn giản. Trong quá trình sử dụng ta có thể cần phải chỉnh các điều kiện và sử dụng kết hợp điều kiện kiểm tra trong trường hợp cần thiết.

2. Lệnh điều kiện nếu-thì mở rộng 1

Lệnh điều khiển nếu-thì mở rộng 1 ngoài cho phép thực thi một cụm lệnh nếu điều kiện kiểm tra là đúng còn cho phép thực thi một cụm lệnh khác nếu điều kiện kiểm tra sai. Điều kiện này có thể thấy rõ trong ví dụ về quyết định hành động khi mua vé xổ số ở đầu của bài học này. Lệnh này là lệnh mở rộng hơn từ lệnh điều khiển nếu-thì đơn giản ở trên. Như vậy với lệnh này chúng ta cần 3 tập lệnh con: Thứ nhất là lệnh kiểm tra điều kiện, thứ hai là tập lệnh thực thi khi điều kiện kiểm tra là đúng và thứ ba là tập lệnh thực thi khi điều kiện kiểm tra là sai.



Hình 4.9: Cấu trúc lệnh điều kiện nếu-thì mở rộng 1

Ví dụ 5: Ở bài học về lệnh điều khiển nếu-thì đơn giản có một bài toán đơn giản là cảnh báo ra còi nếu nhiệt độ môi trường lớn hơn 35°C. Với ví dụ này ta thay đổi tín hiệu cảnh báo còi bằng đèn. Cụ thể như sau: Thực hiện đo nhiệt độ môi trường và bật đèn cảnh báo nếu nhiệt độ cao hơn 35°C và tắt đèn khi ngược lại.

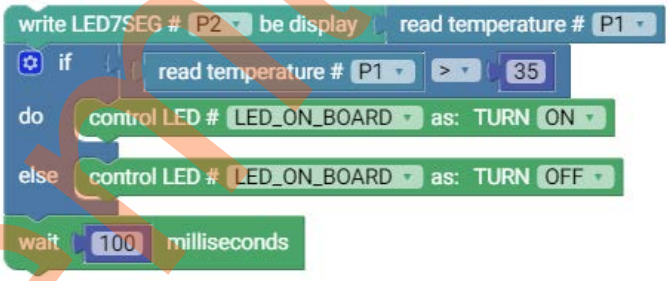
Giả sử ta vẫn sử dụng chương trình từ phần trên và thay cụm lệnh bật còi cảnh báo bằng lệnh bật đèn, chúng ta sẽ có chương trình như sau:

Hình 4.10: Chương trình cảnh báo nhiệt độ cao bằng tín hiệu đèn chưa hoàn thiện

Chương trình này được bổ sung thêm lệnh hiển thị nhiệt độ lên màn hình hiển thị số nhằm dễ dàng quan sát và kiểm nghiệm. Khi tiến hành thử nghiệm chương trình này, ta thấy ban đầu khi nhiệt độ dưới 35°C thì đèn ở trạng thái tắt đến khi nhiệt độ tăng và vượt ngưỡng 35°C thì đèn sẽ sáng đúng như khi yêu cầu bài toán. Nhưng sau khi đèn đang ở trạng thái sáng mà nhiệt độ giảm xuống dưới 35°C thì đèn sẽ vẫn tiếp tục sáng mà không tắt đi. Điều này không phù hợp và sau lần bật đèn đầu tiên thiết bị không còn vai trò cảnh báo nữa. Vậy lý do ở đây là gì?

Ta thấy trong chương trình có lệnh đưa đèn LED lên trạng thái bật (ON) nhưng không có bất cứ một lệnh nào để tắt đèn, đây chính là lý do tại sao sau khi được bật lên, nhiệt độ đã giảm xuống mà đèn trên thiết bị vẫn sáng.

Để giải quyết vấn đề này cần phải bổ sung vào chương trình một đoạn lệnh để thực hiện tắt đèn LED. Đoạn lệnh này được chạy khi nhiệt độ môi trường nhỏ hơn hoặc bằng 35°C. Sử dụng lệnh điều khiển nếu-thì mở rộng 1 và xây dựng chương trình như sau:



Hình 4.11: Chương trình cảnh báo nhiệt độ cao bằng tín hiệu đèn hoàn thiện

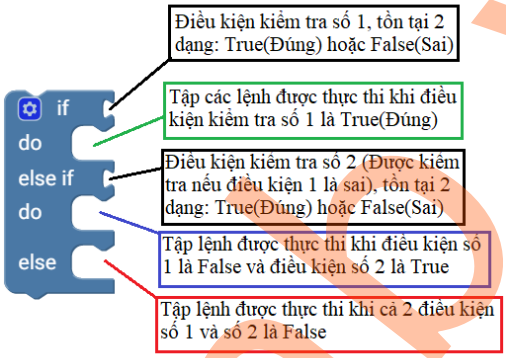
Ngoài lệnh dùng để bật đèn LED, chương trình đã được bổ sung một lệnh để tắt đèn LED vào vị trí khi điều kiện kiểm tra là sai (Khi nhiệt độ nhỏ hơn hoặc bằng 35°C). Cụ thể, chương trình được phân tích như sau:

- Chương trình đọc nhiệt độ từ môi trường và so sánh với 35.
- Nếu nhiệt độ đọc được lớn hơn 35°C thì điều kiện kiểm tra trả về là **True**(Đúng), tiến hành thực thi lệnh bật đèn (Control ON), bỏ qua lệnh tắt đèn và thoát khỏi lệnh điều khiển nếu-thì để thực hiện các lệnh tiếp theo.
- Nếu nhiệt độ nhỏ hơn hoặc bằng 35°C thì điều kiện kiểm tra là **False**(Sai), tiến hành bỏ qua lệnh bật đèn và thực thi lệnh tắt đèn (Control OFF) sau đó thoát khỏi lệnh điều khiển nếu-thì và thực hiện lệnh tiếp theo.

- Chương trình thực thi như vậy đảm bảo khi nhiệt độ thay đổi từ lớn hơn 35°C về nhỏ hơn hoặc bằng 35°C thì lệnh tắt đèn được thực thi, vì thế tín hiệu cảnh báo nguy hiểm được tắt đi và đảm bảo đúng yêu cầu bài toán mà đã được đặt ra.

Bài toán mở rộng: *Thực hiện điều khiển bật đèn khi ánh sáng môi trường có độ sáng nhỏ hơn 60% và tự động tắt đèn khi độ sáng môi trường lớn hơn hoặc bằng 60%.*
(Gợi ý: Sử dụng cảm biến ánh sáng ở chế độ PERCENT)

3. Lệnh điều kiện nếu-thì mở rộng 2



Hình 4.12: Cấu trúc lệnh điều kiện nếu-thì mở rộng 2

Ở ví dụ về quyết định khi mua vé xổ số chúng ta thấy khi ta giả sử trúng giải ta có rất nhiều lựa chọn khác nhau về giải sẽ trúng. Khi trúng xổ số, ta có thể trúng giải đặc biệt, giải nhất, giải nhì, giải ba… hoặc giải khuyến khích. Với mỗi trường hợp trúng xổ số ta có các hành động xử lý khác nhau vì vậy lệnh điều khiển nếu-thì mở rộng 1 là chưa đủ để đáp ứng yêu cầu bài toán đó. Do đó, xuất hiện lệnh điều khiển nếu-thì mở rộng 2 để đáp ứng và giải quyết những bài toán có nhiều trường hợp điều kiện cần xử lý.

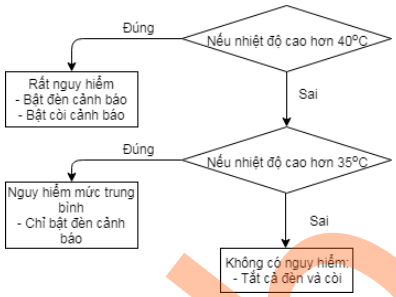
Tiếp tục với bài toán cảnh báo ở trên, trong thực tế việc đưa ra tín hiệu cảnh báo không chỉ dừng lại ở việc đưa ra cảnh báo nguy hiểm hay không nguy hiểm mà có nhiều mức cảnh báo nguy hiểm khác nhau tùy thuộc vào các điều kiệ. Ví dụ như cảnh báo bão chúng ta có 12-15 cấp độ bão thông thường (Có thể lớn hơn trong trường hợp xuất hiện bão mạnh hơn nữa), hoặc cảnh báo cháy rừng chúng ta cũng có 5 mức độ nguy hiểm khác nhau và với mức độ nguy hiểm khác nhau các cơ quan chức năng cần có thái độ và sự chuẩn bị đối phó khác nhau. Bài học này tiếp tục với bài toán cảnh báo cháy rừng, để đơn giản cho ví dụ ta chỉ quan tâm đến nhiệt độ môi trường để thiết kế thiết bị cảnh báo.

Hình 4.13: Các ngưỡng cảnh báo cháy rừng

Ví dụ 6: Thiết kế thiết bị cảnh báo cháy rừng theo thông số nhiệt độ (T°) môi trường. Thiết bị hoạt động theo nguyên tắc sau:

- Cảnh báo bằng đèn và còi nếu nhiệt độ cao hơn 40°C (T° > 40°c)
- Chỉ cảnh báo bằng đèn nếu nhiệt độ cao hơn 35°C và nhỏ hơn hoặc bằng 40°C (35°C<T°≤40°C)
- Không đưa ra cảnh báo trong điều kiện an toàn (35°C≤T°)

Từ yêu cầu bài toán ta xây dựng được một lưu đồ thuật toán như sau:



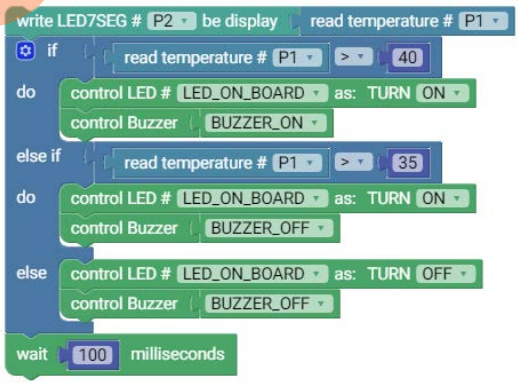
Hình 4.14: Lưu đồ thuật toán chương trình cảnh báo cháy rừng

Ban đầu, ta tiến hành đọc nhiệt độ môi trường và so sánh với 40°C. Nếu nhiệt độ lớn hơn 40°C nghĩa là mức độ nguy hiểm đang ở mức cao thì thực thi lệnh bật đèn, bật còi cảnh bảo tương ứng với mức nguy hiểm cao nhất.

Nếu nhiệt độ không lớn hơn 40°C tức là môi trường đang ở mức nguy hiểm trung bình hoặc không nguy hiểm vì vậy cần tiếp tục kiểm tra để xác định rừng đang ở mức nào trong 2 mức nói trên. Để xác định được điều đó cần tiếp tục so sánh nhiệt độ đọc được với 35°C, nếu nhiệt độ lớn hơn 35°C tức là mức nguy hiểm ở mức trung bình và chỉ cần bật đèn để cảnh báo và để tránh trường hợp còi không tự tắt ta cần bổ sung lệnh tắt cảnh báo còi.

Trường hợp còn lại nhiệt độ môi trường nhỏ hơn hoặc bằng 35°C tức là không có nguy hiểm, rừng ở trạng thái an toàn về cảnh báo cháy rừng vậy cần tiến hành tắt cả còi và đèn cảnh báo.

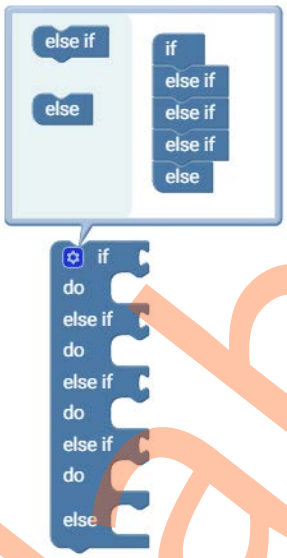
Chương trình được xây dựng phù hợp với yêu cầu bài toán có nội dung như sau:



Hình 4.15: Chương trình cảnh báo cháy rừng 3 mức

Tiến hành nạp chương trình vào Board và chạy thử nghiệm ta thấy thiết bị hoạt động đúng như yêu cầu đã đặt ra.

Chúng ta có thể tiếp tục sử dụng các lệnh điều khiển nếu-thì mở rộng hơn nữa để điều khiển chương trình ở nhiều chế độ khác nhau tùy thuộc vào yêu cầu thực tế. Để sử dụng lệnh này mở rộng hơn nữa ta sử dụng thêm các khối **elseif** hoặc **else** trong cấu hình của lệnh của lệnh điều khiển nếu-thì cho phù hợp với số lượng trường hợp mà chúng ta có. Dưới đây là một cấu trúc có thể tham khảo:



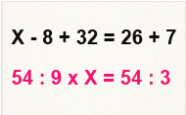
Hình 4.16: Cấu trúc lệnh điều kiện nếu-thì mở rộng tổng quát

Về phương thức hoạt động lệnh này hoạt động tương đương với lệnh điều khiển điều kiện nếu-thì 2 với số lượng trường hợp cần quan tâm nhiều hơn. Trong thực tế khi cần thiết, ta có thể linh động sử dụng các hàm này để phù hợp với chương trình.

Bài toán mở rộng: *Thực hiện thiết bị cảnh báo về mức độ ánh sáng có nguyên tắc hoạt động như sau: Sử dụng cảm biến ánh sáng để đo ánh sáng môi trường, thể hiện mức độ sáng tương ứng thông qua độ sáng đèn LED:*

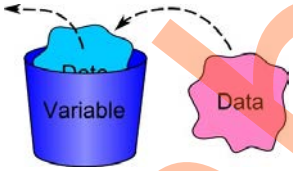
- *Với điều kiện ánh sáng nhỏ hơn 30%, điều khiển đèn sáng ở mức 0%.*
- *Với điều kiện ánh sáng từ 30%-60%, điều khiển đèn sáng ở mức 40%.*
- *Với điều kiện mức sáng từ 60%-90%, điều khiển đèn sáng ở mức 80%.*
- *Với điều kiện mức sáng từ 90-100%, điều khiển đèn sáng ở mức 100%.*

BÀI 5: BIẾN (VARIABLE)



Trên đây là một hình ảnh mà chúng ta thường xuyên gặp trong môn toán và công việc của chúng ta là phải giải để tìm ra giá trị của "X". "X" là một đại diện cho một số nào đó mà chúng ta chưa biết giá trị và ta cần tìm giá trị cho nó. Trong lập trình các "X" như thế được gọi là các biến.

Biến là các đối tượng sử dụng để chứa giá trị nào đó và, giá trị đó có thể thay đổi được trong suốt quá trình thực hiện chương trình.



Hình 5.1: Ví dụ về biến

Trên đây là hình ảnh ví dụ về biến. Biến ở đây được đại diên bởi cái xô (chậu). Chúng ta có thể dùng xô để chứa chất lỏng như nước, sơn, dầu,... hoặc có thể ngay cả những thứ không phải là chất lỏng. Và lượng chất lỏng hoặc thứ được chứa trong xô có thể thay đổi trong suốt quá trình chúng ta sử dụng. Biến cũng vậy, nó là một đối tượng dùng để mang giá trị, giá trị của biến có thể thay đổi trong suốt quá trình sử dụng nếu cần thiết.

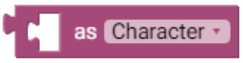
Các loại dữ liệu mà biến có thể chứa trong CloverBlock được mô tả trong bảng sau:

Kiểu dữ liệu	Chú thích
Boolean	Là kiểu dữ liệu sử dụng cho các đối tượng chỉ tồn tại 2 giá trị như: True/False (Đúng/Sai), On/Off(Bật/Tắt),...
Số nguyên (Short number, Number, Large number)	Là kiểu sử dụng cho các đối tượng thể hiện dưới dạng số nguyên. Có nhiều dạng như: Số nguyên ngắn, Số nguyên trung bình và số nguyên dài. Ví dụ: 1234, -4321,…
Số thực (Decimal)	Là kiểu dữ liệu sử dụng cho các đối tượng có giá trị là số thực. Ví dụ: 12,34; -20,18;…
Ký tự (Character)	Là kiểu dữ liệu sử dụng cho các đối tượng có giá trị là các ký tự như: 'A', 'B',...,'0', '1',...
Chuỗi văn bản (Text)	Là kiểu dữ liệu sử dụng cho các đối tượng có giá trị là tổ hợp của nhiều ký tự ghép lại. Ví dụ: "Hello"

Bảng 5.1: Các kiểu dữ liệu của CloverBlock

Những kiểu dữ liệu trên cũng là những kiểu dữ liệu cơ bản và thường được sử dụng trong các ngôn ngữ lập trình khác như C, Java, Python...

Về các lệnh liên quan đến biến bao gồm các lệnh sau:

Tên lệnh	Lệnh	Vị trí	Chú thích
Lấy giá trị biến		Variables	Lấy giá trị của biến "**item**" sử dụng làm đầu vào cho các lệnh khác. Chú ý việc sử dụng giá trị của biến yêu cầu có kiểu dữ liệu phù hợp với kiểu dữ liệu mà lệnh sử dụng yêu cầu.
Ghi giá trị biến		Variables	Ghi giá trị của biến "**item**" bằng bằng một giá trị khác. Giá trị truyền vào cho biến trong cả chương trình nên đồng nhất là một kiểu dữ liệu.
Đọc biến dưới dạng kiểu dữ liệu khác		Variables	Thực hiện chuyển đổi kiểu dữ liệu của biến muốn đọc sang kiểu dữ liệu khác rồi mới truyền cho hàm sử dụng giá trị của biến này.

Bảng 5.2: Các lệnh liên quan đến biến của CloverBlock

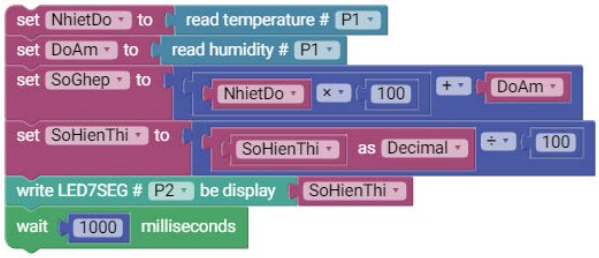
Ví dụ 1: Thực hiện đọc nhiệt độ và độ ẩm của môi trường và hiển thị lên màn hình hiển thị số theo yêu cầu cụ thể như sau:

- Nhiệt độ và độ ẩm cùng được hiển thị lên màn hình hiển thị số
- Mỗi thông số chiếm 02 ô hiển thị trên màn hình. Nhiệt độ nằm phía bên trái và độ ẩm nằm phía bên phải
- Hai thông số được ngăn cách nhau bởi dấu chấm
- Sử dụng biến để hoàn thành chương trình

Phân tích:

- Sử dụng 2 biến để chứa giá trị nhiệt độ và độ ẩm đã đọc được từ môi trường.
- Yêu cầu là đưa hai thông số lên cùng 1 màn hình hiển thị tuy nhiên hàm hiển thị lên màn hình chỉ cho phép điều khiển hiển thị 1 số lên cả màn hình vì vậy cần chuyển 2 số này về 1 số duy nhất rồi mới hiển thị.
- Vậy đưa 2 số về 1 số bằng cách nào mà vẫn có thể quan sát độc lập? Yêu cầu của bài toán là mỗi thông số chiếm 02 ô hiển thị trên màn hình, nhiệt độ nằm về phía bên trái và độ ẩm nằm về phía bên phải. Từ yêu cầu này có thể sử dụng phép nhân và phép cộng kết hợp để ghép 2 số.
  Ví dụ: Nhiệt độ:12 và Độ ẩm: 34. Ta có: 12x100 = 1200 → 1200 + 34 = 1234.
- Yêu cầu 2 số ngăn cách nhau bởi dấu chấm. Tới đây có thể thực hiện chia số vừa ghép được cho 100. Chúng ta có: 1234:100 = 12.34. Đây chính là cách chúng ta giải quyết yêu cầu này.
- Tuy nhiên, kiểu dữ liệu của cả nhiệt độ và độ ẩm đều là số nguyên trong khi số có dấu chấm (,) là số thực. Vì vậy, trước khi thực hiện phép chia ta cần sử dụng phép chuyển đổi số liệu sang kiểu số thực sau đó thực hiện phép chia để đạt được kết quả chính xác.

Từ những phân tích trên ta xây dựng được chương trình như sau:



Hình 5.2: Chương trình hiển thị đồng thời nhiệt độ và độ ẩm

Bước 1: Sử dụng 2 biến *NhietDo* và *DoAm* để lưu trữ thông tin Nhiệt độ và Độ ẩm đọc được từ cảm biến để phục vụ tính toán.

Bước 2: Biến *SoGhep* được dùng để kết hợp hai biến *NhietDo* và *DoAm* thành một thông số bằng cách nhân *NhietDo* với 100 rồi cộng với *DoAm* đảm bảo 2 số được hiển thị độc lập.

Bước 3: Thêm dấu chấm để ngăn cách 2 thông số bằng cách chuyển đổi biến *SoGhep* về kiểu số thực sau đó chia cho 100 để xuất hiện dấu chấm ngăn cách và đưa vào biến *SoHienThi*.

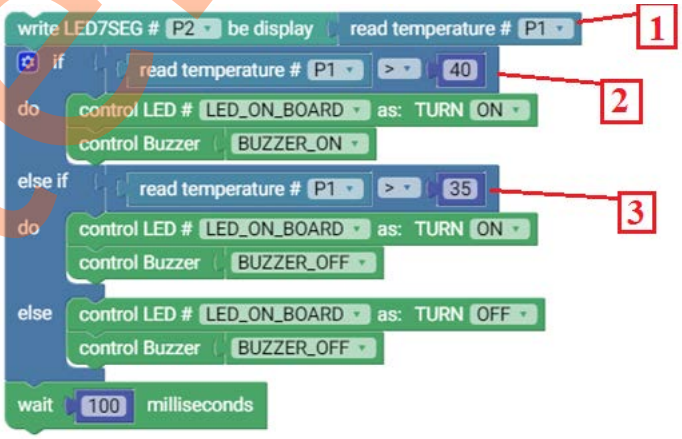
Bước 4. Sử dụng biến *SoHienThi* làm đầu vào cho lệnh điều khiển hiển thị lên màn hình hiển thị số và tạm dừng 1 giây trước khi thực hiện lần cập nhật thông số tiếp theo.

Ví dụ 2: Quay lại với bài toán ở ví dụ số 6 của bài học số 4.

Thiết kế thiết bị cảnh báo cháy rừng theo thông số nhiệt độ môi trường được hoạt động theo nguyên tắc sau:

- Cảnh báo bằng đèn và còi nếu nhiệt độ cao hơn 40°C
- Chỉ cảnh báo bằng đèn nếu nhiệt độ cao hơn 35°C và nhỏ hơn hoặc bằng 40°C
- Không đưa ra cảnh báo trong điều kiện an toàn (Nhiệt độ nhỏ hơn hoặc bằng 35°C)

Với các yêu cầu bài toán và phân tích ta đã thiết kế được một chương trình như sau:



Hình 5.3: Chương trình của ví dụ 6 - Bài học số 4

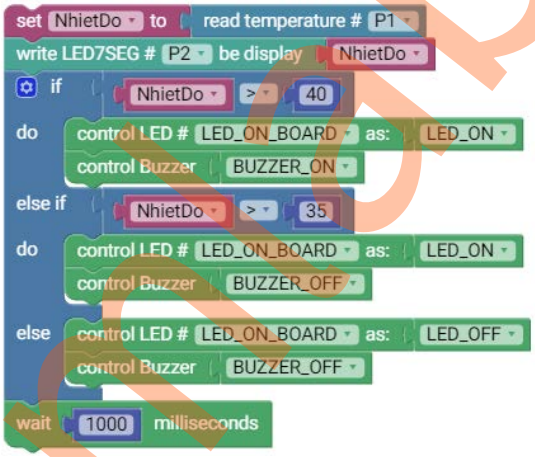
Ở chương trình này có 3 lệnh đọc nhiệt độ: 1 lần cho hiển thị lên màn hình, một lần để so sánh với ngưỡng cao nhất là 40 và lần đọc cuối cùng để so sánh với ngưỡng thấp là 35. Tuy

nhiên, vấn đề xảy ra là các cảm biến đều có sai số và không hoàn toàn ổn định giữa các lần đọc. Vì thế nhiệt độ đọc được có thể sai khác giữa các lần đọc ở cùng điều kiện.

Ví dụ nhiệt độ đang là 40°C nhưng giá trị trả về cảm biến có thể là 39, 40 hoặc 41. Với chương trình đã thiết kế, giả sử nhiệt độ ở lần đầu đọc được là 40°C nhưng ở lần thứ hai có thể là 41°C nên có thể trong khi chạy chương trình trên xuất hiện tình huống: Trên màn hình hiển thị nhiệt độ là 40°C nhưng thấy cả còi và đèn đều ở mức cảnh báo (Vì lần đọc thứ 2 đọc được 41°C) gây cho hiểu lầm rằng chương trình có vấn đề về logic hoặc thuật toán trong khi chương trình vẫn đảm bảo hoạt động đúng. Đây chính là tính thiếu đồng nhất thông số. Ở trường hợp này ta có thể sử dụng biến để giải quyết vấn đề. Giải pháp thực hiện cụ thể như sau:

- Đọc giá trị nhiệt độ vào 1 biến (Chỉ đọc một lần cho mỗi lần tiến hành kiểm tra).
- Sử dụng biến lưu giá trị nhiệt độ đọc được để hiển thị lên màn hình.
- Sử dụng biến lưu giá trị nhiệt độ đọc được để tiến hành so sánh tìm ra mức cảnh báo phù hợp.

Chương trình để khắc phục vấn đề trên được xây dựng lại như sau:

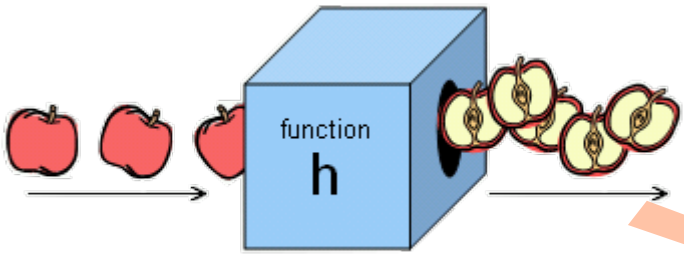


Hình 5.4: Chương trình ví dụ về dùng biến đồng nhất giá trị

Chương trình này đã được khắc phục được vấn đề thiếu tính đồng bộ như đã nhắc ở trên do biến chỉ được cập nhật một lần và được sử dụng nhiều lần để hiển thị, so sánh,... Do trong suốt quá trình này giá trị của biến NhietDo không thay đổi và không chịu ảnh hưởng của sự sai số so linh kiện như các thiết bị cảm biến nên các thông số được đồng nhất giữa hiển thị, so sánh hoặc sử dụng ở các lệnh khác.

Biến có thể được sử dụng để hạn chế tính thiếu đồng nhất với những trường hợp giá trị đầu vào từ môi trường. Nếu thông số đầu vào được sử dụng nhiều hơn một lần ta nên thực hiện chỉ đọc một lần vào biến sau đó sử dụng biến đó đại diện cho thông số mà chương trình cần để đối chiếu. Giá trị của biến chỉ được cập nhật 1 lần cho mỗi một lượt khảo sát.

BÀI 6: HÀM, THỦ TỤC (FUNCTION)



Trên đây là một hình ảnh ví dụ về hàm. Ở hình ảnh trên, ta thấy bên trái là các trái táo và bên phải là các trái táo đã được bổ làm hai. Ở giữa là một khối lập phương với chức năng đưa táo ở đầu vào và đưa ra là dạng táo đã được bổ. Ta có thể đoán hình lập phương này là đại diện cho máy bổ táo (Chức năng chính là đưa táo từ dạng cả quả về dạng đã được bổ). Giả sử, ở một vùng quê nào đó trồng được rất nhiều táo và công việc bổ táo là công việc quan trọng mà gia đình nào cũng phải làm trước khi đưa táo đi chế biến thành phẩm để bán. Do số lượng công việc lớn nên gia đình nào cũng muốn có một máy bổ táo của riêng mình để giảm sức lao động của con người. Tuy nhiên, liệu có phải tất cả các hộ gia đình đều cần phải biết nguyên lý hoạt động, cấu tạo và cách chế tạo máy bổ táo? Điều này là không cần thiết. Chúng ta chỉ cần một hoặc một số đơn vị, công ty biết về cách chế tạo máy bổ táo còn việc của các hộ gia đình là mua về, sử dụng và không cần thiết phải biết về cấu trúc bên trong của chiếc máy này. Ta có thể thấy cấu trúc của máy bổ táo chỉ cần được xây dựng một lần và nhân bản thành nhiều thiết bị phục vụ đến người dùng với cấu trúc đã được đóng kín và lúc này người sử dụng chỉ cần quan tâm đến tính năng và cách sử dụng của chiếc máy đó.

Ví dụ trên là một hình ảnh thực tế tương đương với hàm, thủ tục trong lập trình. Trong khi thiết kế các chương trình sử dụng CloverBlock hay bất kỳ các ngôn ngữ lập trình nào khác sẽ thường xuyên xuất hiện các đoạn chương trình (tổ hợp nhiều câu lệnh) được sử dụng lặp lại ở nhiều nơi trong chương trình. Có một cách đơn giản là chúng ta sao chép các lệnh vào vị trí mà chúng ta cần sử dụng. Điều này gây lãng phí của chúng ta rất nhiều thời gian đặc biệt đối với các đoạn chương trình được lặp lại có nhiều câu lệnh. Ngoài ra, trong quá trình thiết kế nếu có chỉnh sửa chúng ta phải tiến hành sửa tại tất cả các vị trí đang sử dụng cụm lệnh đó gây lãng phí thời gian và có thể có những thiếu sót khó tránh khỏi.

Vậy hàm, thủ tục là gì? Có thể giải thích tóm gọn định nghĩa về hàm (thủ tục) như sau:

- Hàm là một khối lệnh được xây dựng để thực hiện một chức năng cụ thể nào đó.
- Hàm luôn luôn được xác định bởi một tên gọi cụ thể. Mọi thao tác với hàm đều sẽ được thực hiện thông qua tên gọi này.

Ví dụ: Trong giới thiệu mở đầu về máy bán táo: Hàm được cấu thành bởi các chi tiết máy sắp đặt theo một cấu trúc xác định nào đó với chức năng bổ trái táo. Hàm này được đặt tên là "Máy bổ táo" và khi cần sử dụng chức năng bổ táo chúng ta chỉ cần đưa táo vào chiếc máy này mà không cần thêm thao tác nào khác.

Thông thường, chúng ta sẽ có 2 dạng cơ bản:

- Hàm: sau khi thực hiện một chức năng cụ thể, hàm trả ra bên ngoài một giá trị nào đó.
- Thủ tục: là một trường hợp của hàm sau khi thực hiện một chức năng cụ thể nhưng không trả về giá trị nào cả.

Thực tế, rất nhiều lệnh được xây dựng trong CloverBlock là một hàm, thủ tục nào đó đã được đóng gói thành một lệnh giúp người học nhanh chóng tiếp cận lập trình.

Ví dụ:

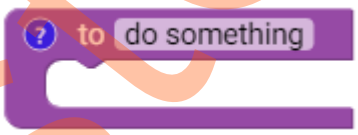
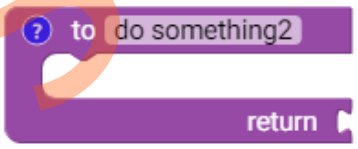
Thủ tục: Lệnh hiển thị một số lên màn hình hiển thị số.

Đây là một lệnh đại diện cho thủ tục có vai trò thực hiện gửi số ra màn hình hiển thị. Trong lệnh này bao gồm rất nhiều lệnh nhỏ khác được sắp xếp phù hợp để gửi số tương ứng ra màn hình. Lệnh này sau khi thực hiện xong không thực hiện trả về giá trị nào cả. *(Không có trả về giá trị nên gọi là thủ tục).*

Hàm: Lệnh đọc giá trị nhiệt độ từ cảm biến.

Lệnh này có vai trò đọc giá trị nhiệt độ từ cảm biến nhiệt độ, độ ẩm được kết nối với Board điều khiển. Tương tự như lệnh điều khiển hiển thị số lên màn hình lệnh này cũng được tạo thành từ rất nhiều lệnh khác. Và sau khi thực hiện xong lệnh này thực hiện trả về giá trị của nhiệt độ đã đọc được. *(Có trả về giá trị là nhiệt độ nên gọi là hàm).*

Ngoài các hàm thủ tục đã được CloverBlock xây dựng sẵn dưới dạng lệnh người dùng hoàn toàn có thể thiết kế các hàm, thủ tục để phục vụ riêng cho chương trình của mình trong trường hợp cần thiết. Các lệnh về xây dựng hàm và thủ tục được liệt kê trong bảng sau:

STT	Block	Vị trí	Chức năng
1		Functions	Xây dựng thủ tục có tên là ***"do something"***.
2		Functions	Xây dựng hàm có tên là ***"do something2"***.

Bảng 6.1: Các lệnh về xây dựng hàm, thủ tục

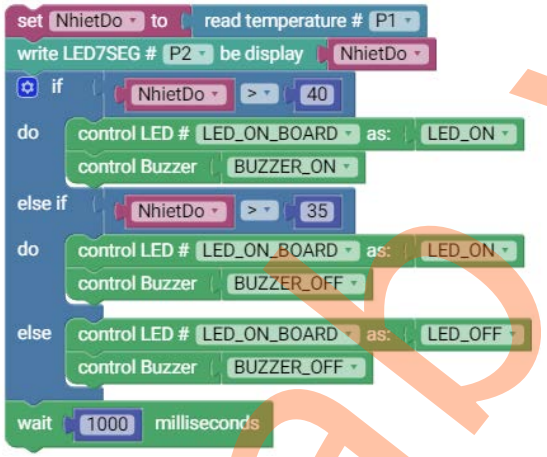
1. Thủ tục (Hàm không trả về dữ liệu)

Ví dụ 1: Tiếp tục với ví dụ số 2 của bài học số 5.

Thiết kế thiết bị cảnh báo cháy rừng theo thông số nhiệt độ môi trường được hoạt động theo nguyên tắc sau:

- Cảnh báo bằng đèn và còi nếu nhiệt độ cao hơn 40°C
- Chỉ cảnh báo bằng đèn nếu nhiệt độ cao hơn 35°C và nhỏ hơn hoặc bằng 40°C
- Không đưa ra cảnh báo trong điều kiện an toàn (Nhiệt độ nhỏ hơn hoặc bằng 35°C)

Với các phân tích chúng ta đã có chương trình được xây dựng như sau:



Hình 6.1: Chương trình của ví dụ 2 – Bài học số 5

Ở chương trình trên ta thấy có 3 loại cảnh báo:

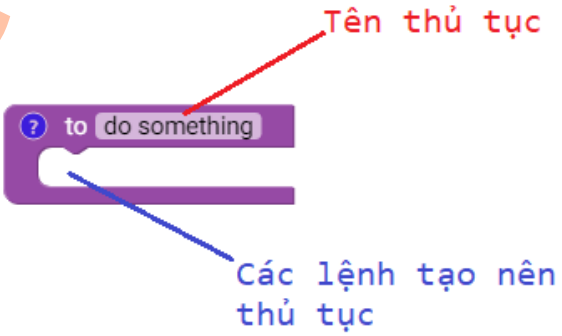
Nguy hiểm cao: Bật cả đèn và còi.

Nguy hiểm thấp: Chỉ bật đèn, không bật còi.

Không nguy hiểm: Tắt cả đèn và còi.

Với 3 mức cảnh báo chúng ta có các tín hiệu cảnh báo khác nhau được quy định bằng đèn và còi. Tuy nhiên, trong quá trình thiết kế cần thay đổi các tín hiệu cảnh báo bằng đèn và còi sang dạng tín hiệu khác, khi đó chúng ta sẽ tốn nhiều thời gian dò tìm đoạn chương trình thực hiện cảnh báo để chỉnh sửa. Vì vậy, chúng ta có thể xây dựng thủ tục cho 3 loại cảnh báo này. Sau đó, tại các vị trí cảnh báo tương ứng chúng ta chỉ cần sử dụng các lệnh tương ứng với các thủ tục đã được xây dựng trước đó.

Chi tiết về lệnh xây dựng thủ tục như sau:



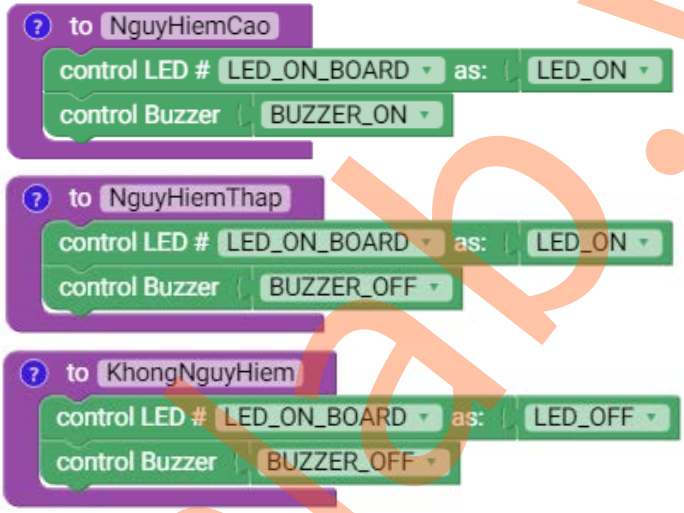
Hình 6.2: Chi tiết về lệnh tạo thủ tục

Tên thủ tục là tên mà chúng ta đặt cho thủ tục muốn xây dựng ví dụ như: "NguyHiemCao", "NguyHiemThap", "KhongNguyHiem"… Tên này do người sử dụng tùy đặt.

Chú ý: Không sử dụng một tên cho nhiều hơn 1 một thủ tục, hàm được xây dựng. Tên của thủ tục, hàm nên được đặt gợi nhớ đến chức năng sử dụng của thủ tục, hàm đó.

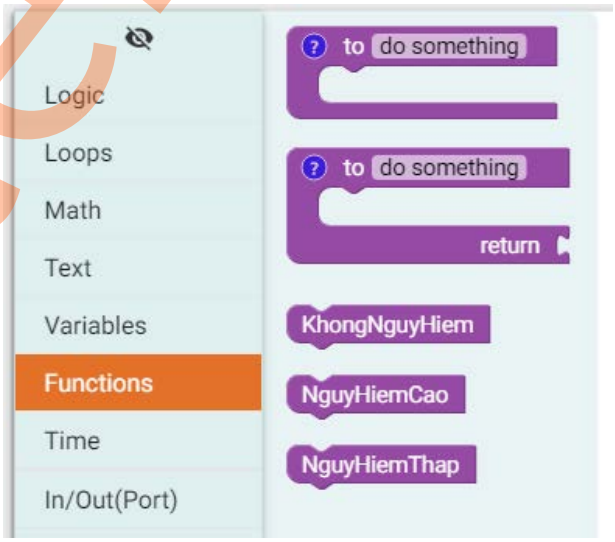
Các lệnh tạo nên thủ tục, hàm là tập hợp các lệnh sử dụng để tạo thành thủ tục, hàm muốn xây dựng. Số lượng lệnh trong một thủ tục ít hay nhiều phụ thuộc vào người thiết kế chương trình và vai trò của thủ tục, hàm đó.

Chúng ta tiến hành xây dựng lần lượt các thủ tục cảnh báo tương ứng với 3 mức độ đã phân tích ở trên như sau:



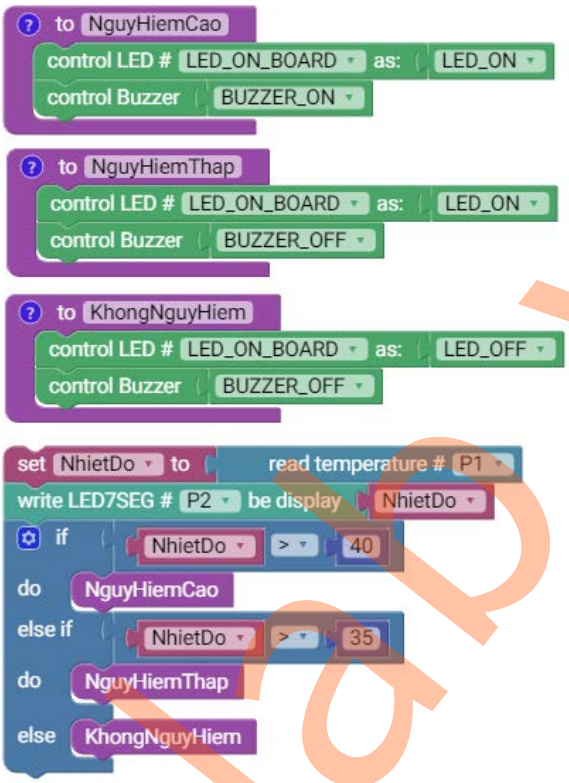
Hình 6.3: Các thủ tục cảnh báo được xây dựng

Sau khi đã xây dựng xong các thủ tục, để xây dựng chương trình tương đương chương trình ban đầu chúng ta tiến hành thay thế các cụm lệnh cảnh báo bằng cách thủ tục tương ứng. Các lệnh tương đương với các thủ tục, hàm tương ứng được xuất hiện ở cụm Block Functions như sau:



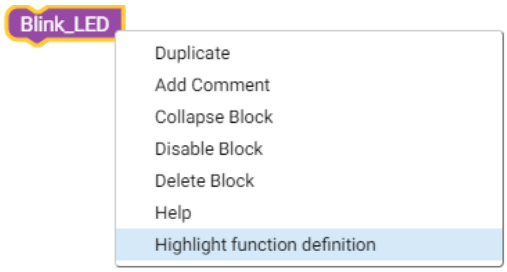
Hình 6.4: Sử dụng các hàm, thủ tục đã được xây dựng

Để sử dụng các thủ tục, hàm này trong chương trình, chúng ta chỉ cần sử dụng các Block có tên tương ứng. Sau khi tiến hành thay thế các cụm lệnh cảnh báo bằng cách sử dụng các Block tương ứng với thủ tục đã xây dựng ở trên, chúng ta có chương như sau:



Hình 6.5: Chương trình sử dụng thủ tục đã xây dựng

Chương trình trên đã được triển khai dưới dạng thủ tục và chúng ta thấy chương trình trở nên dàng đọc hiểu hơn chương trình ban đầu. Nhìn vào đoạn chương trình chính chúng ta thấy rõ ràng khi nhiệt độ cao hơn 40°C thì lệnh cảnh báo mức nguy hiểm cao được thực thi, nếu nhỏ hơn hoặc bằng 40°C nhưng lớn hơn 35°C thì lệnh cảnh báo mức nguy hiểm thấp được thực thi và còn lại nếu không lớn hơn 35°C thì lệnh cảnh báo không nguy hiểm được thực thi. Về cụ thể các lệnh cảnh báo trên như thế nào chúng ta có thể dễ dàng xem xét và tìm hiểu bằng cách tìm đến nội dung thủ tục, hàm tương ứng hoặc sử dụng tính năng **"Highlight function definition"** để có thể nhanh chóng tìm được nội dung của hàm, thủ tục cần xem xét.



Hình 6.6: Tính năng Highlight function definition

2. Hàm (Có trả về giá trị)

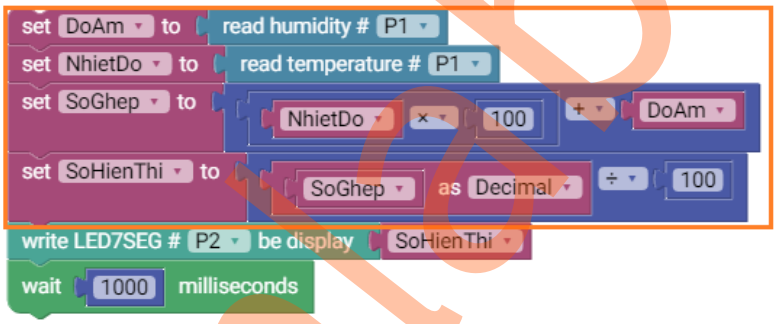
Từ ví dụ số 1 chúng ta đã thấy được ưu điểm của việc sử dụng việc xây dựng thủ tục, hàm trong chương trình. Phần này tiếp tục giới thiệu về cách xây dựng hàm (có trả về giá trị). Khác với thủ tục, hàm sau khi thực hiện toàn bộ các lệnh sẽ trả về một giá trị nào đó có thể được cung cấp để sử dụng cho một lệnh khác.

Ví dụ 2: Tiếp tục với ví dụ số 1 của bài học số 5.

Thực hiện đọc nhiệt độ và độ ẩm của môi trường và hiển thị lên màn hình hiển thị số theo yêu cầu cụ thể như sau:

- Nhiệt độ và độ ẩm cùng được hiển thị lên màn hình hiển thị số
- Mỗi thông số chiếm 02 ô hiển thị trên màn hình. Nhiệt độ nằm phía bên trái và độ ẩm nằm phía bên phải
- Hai thông số được ngăn cách nhau bởi dấu chấm

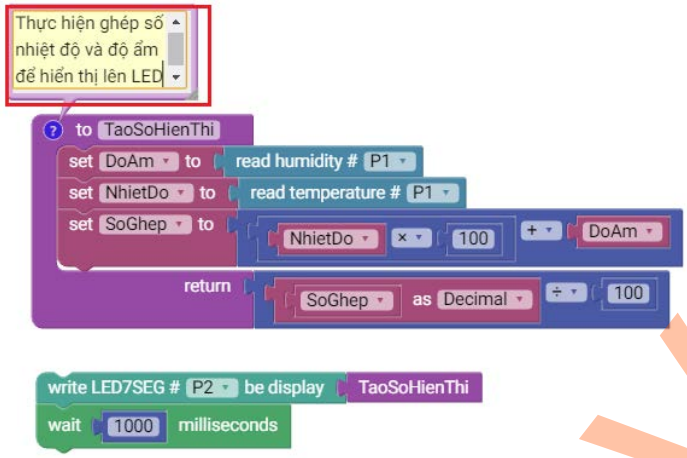
Từ những phân tích từ bài học số 5 chúng ta đã xây dựng được một chương trình đáp ứng được yêu cầu của đề bài như sau:



Hình 6.7: Chương trình của ví dụ 1 – Bài học số 5

Chúng ta thấy đoạn chương trình được khoanh là một đoạn chương trình khá khó hiểu nhất là đối với người đọc chương trình của chúng ta hoặc sau một thời gian dài chúng ta sử dụng lại chương trình để tiếp tục chỉnh sửa và nâng cấp. Vì vậy, chúng ta sẽ thử tạo hàm cho cả khối lệnh trên và đặt cho khối lệnh này một tên gợi nhớ đến vai trò như: "Tạo số hiển thị" hoặc chi tiết hơn là "Tạo số từ nhiệt độ và độ ẩm" để khi đọc lại chương trình chúng ta có thể dễ dàng nắm được vai trò của của cả đoạn lệnh trên từ đó dễ dàng hiểu được chương trình hơn.

Chương trình được xây dựng sử dụng hàm được tạo ra như sau:



Hình 6.8: Chương trình xây dựng và sử dụng hàm

Chúng ta thấy hàm "TaoSoHienThi" được xây dựng tương đương từ các lệnh từ chương trình ban đầu nhưng thay vì đưa giá trị hiển thị vào biến "SoHienThi" thì ở đây giá trị được xây dựng là giá trị trả về của hàm "TaoSoHienThi". Vì vậy, mỗi khi lệnh "TaoSoHienThi" được thực thi sẽ trả về một giá trị là số ghép của nhiệt độ và độ ẩm đã được. Khi đó để hiển thị giá trị yêu cầu lên màn hình hiển thị số chúng ta chỉ cần sử dụng lệnh hiển thị số kết hợp với hàm "TaoSoHienThi" đã được xây dựng ở trên.

Tên hàm, thủ tục hay biến không nên được đặt quá dài do vậy tên này có thể không đủ để diễn tả chức năng của hàm, thủ tục. Khi đó, chúng ta có thể sử dụng tính năng chú thích của CloverBlock và thêm các thông tin chú thích chi tiết hơn vào mỗi hàm, thủ tục để mô tả rõ về chức năng cũng như cách thức hoạt động của hàm, thủ tục. Để thực hiện thêm, sửa các thông tin chú thích chúng ta nhấn chọn vào dấu **"?"** của các lệnh sau đó thêm các thông tin chú thích mong muốn.

Chú ý: Các thông tin chú thích không ảnh hưởng đến quá trình thực hiện, chạy của chương trình được xây dựng.

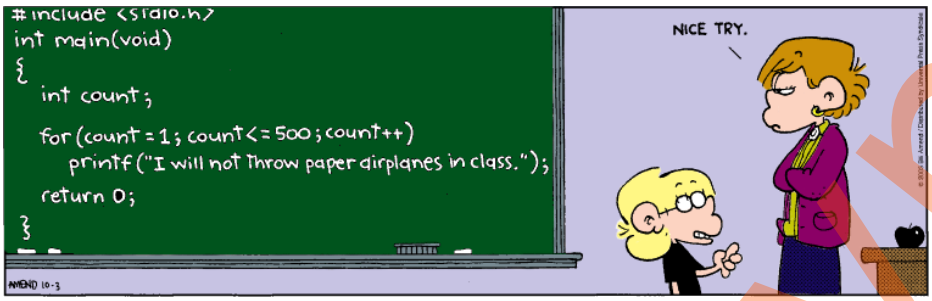
Như vậy, với việc sử dụng hàm, thủ tục chúng ta đã thiết kế được các chương trình trở nên dễ hiểu, dễ dàng chỉnh sửa và ngắn gọn hơn (trong trường hợp các hàm được sử dụng nhiều lần) so với cách xây dựng chương trình không sử dụng hàm và thủ tục.

Ngoài sử dụng các lệnh có sẵn của CloverBlock để tạo thành các hàm chúng ta cũng có thể sử dụng chính các hàm, thủ tục do chúng ta xây dựng để thiết kế, xây dựng các hàm có quy mô phức tạp hơn nếu cần thiết.

Trong quá trình thiết kế chúng ta có thể linh động tạo và sử dụng hàm, thủ tục tại các vị trí cần thiết để đảm bảo chương trình phù hợp vẫn thực hiện đúng yêu cầu và dễ dàng quản lý, chỉnh sửa.

Bài toán mở rộng: *Thiết kế lại một số chương trình đã tìm hiểu xây dựng bằng cách sử dụng xây dựng hàm, thủ tục.*

BÀI 7: LỆNH LẶP XÁC ĐỊNH



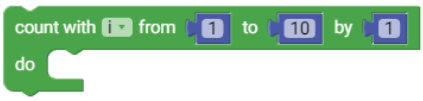
Bài học này giới thiệu về cấu trúc lặp giới hạn. Lặp là một trong những cấu trúc thường gặp trong tất cả các ngôn ngữ lập trình, trong đó có hai kiểu lặp là: Lặp xác định số vòng lặp và Lặp không xác định số vòng lặp. Trong phạm vi bài học này chúng ta dừng lại ở tìm hiểu về cấu trúc lặp xác định số vòng lặp.

Lấy ví dụ về việc thực hiện các phép lặp. Đây là một ví dụ không nhiều bạn thích nhưng là một ví dụ thân thuộc và gần gũi với các bạn học sinh, đó là việc chép phạt. Nếu chúng ta vi phạm một lỗi nào đó đặc biệt là không học bài hoặc không làm bài, có một cách phạt mà các thầy cô hay áp dụng đó là chép phạt. Và kèm với "chiếu chỉ" có của thầy cô sẽ đi cùng một con số nào đó là số lần chúng ta phải chép phạt nội dung đó. Số lần chép phạt càng lớn thì càng là một cơn ác mộng đối với những hưởng hình phạt.Và sau khi lĩnh phạt của chúng ta sẽ phải vừa chép phạt vừa đếm sao cho đủ số lần mà thầy/cô giáo yêu cầu.

Hình 7.1: Ví dụ về lặp xác định

Đây là một ví dụ điển hình cho việc thực hiện vòng lặp. Trong quá trình chép phạt chúng ta hình thành trong đầu hoặc bằng các con số ghi trên bản chép phạt để nhắc nhở về tiến trình được thực hiện và ngay khi thực hiện đủ số lần đó chúng ta dừng lại và thực hiện một công việc khác.

Trong lập trình cũng có những công việc tương tự như thế và đó là các lệnh lặp với số lần xác định. Dưới đây là bảng chi tiết các lệnh liên quan đến lặp số lần xác định:

STT	Block	Vị trí	Chức năng
1		Loop	Thực hiện khối lệnh thực thi bên trong block lặp với số lần được ghi trong ô xanh dương (Ví dụ ở đây là 10 lần)
2		Loop	Thực hiện khối lệnh thực thi bên trong block lặp với số lần xác định thông qua biến **"i"**
3		Loop	Lệnh điều khiển chương trình lặp, bao gồm 2 lựa chọn: - **Break out**: Thực hiện thoát khỏi vòng lặp ngay lập tức. - **Continue with...**: Bỏ qua các lệnh thực thi phía sau và chạy vòng lặp mới.

Bảng 7.1: Các lệnh lặp xác định

1. Lệnh lặp xác định cơ bản.

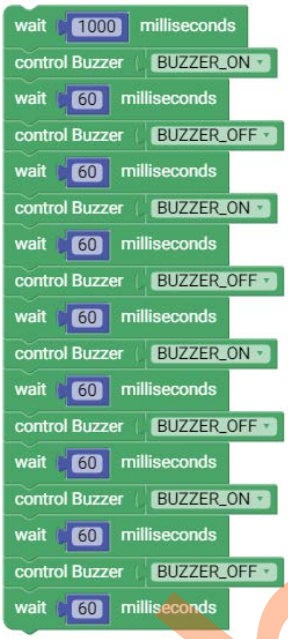
Ví dụ 1: Xây dựng chương trình tạo nhạc âm báo thức.

Đồng hồ báo thức là một thiết bị thân thuộc với hầu hết tất cả mọi người. Mặc dù, gần đây với việc xuất hiện ngày càng nhiều của điện thoại thì điện thoại đã thay thế luôn vai trò của máy báo thức và với điện thoại thông minh chúng ta cũng có rất nhiều âm báo thức khác nhau. Nhưng có một âm báo thức đã trở thành "huyền thoại" mà hiện nay mặc dù có nhiều lựa chọn nhiều người vẫn sử dụng. Ở ví dụ này, chúng ta sẽ thiết kế âm nhạc chuông đó. Nguồn âm thanh có thể tham khảo tại website sau: https://tinyurl.com/ya97ef6m

Một chu kỳ âm báo thức này có thể phân tích thành được giản đồ như sau:

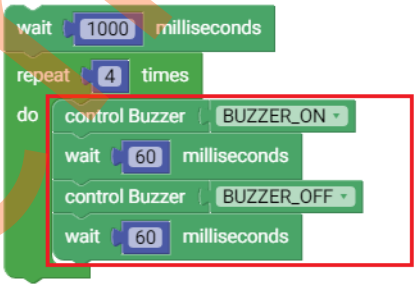
Hình 7.2: Giản đồ của chương trình tạo âm báo thức

Một cách đơn giản để thiết kế chương trình tạo âm thanh báo thức trên bằng CloverBlock là thiết kế tuần tự đúng theo sơ đồ như trên. Chúng ta có chương trình như sau:



Hình 7.3: Chương trình tạo âm báo thức không sử dụng lặp

Tiến hành nạp chương trình trên vào Board và âm báo thức được tạo ra từ còi của Board. Tuy nhiên, chúng ta thấy rằng chương trình trên tương đối dài. Chỉ với 4 chu kỳ bật và tắt liên tục chương trình đã trở nên dài và cồng kềnh. Nếu có một chương trình hoặc bài toán nào đó yêu cầu đến 100 lần chu kỳ như thế này thì chắc chắn sẽ gây rất nhiều khó khăn cho quá trình lập trình. Vì vậy ta cần sử dụng đến sự ưu việt của vòng lặp. Một chu kỳ nhạc âm báo thức bao gồm 4 chu kỳ bật tắt liên tiếp 60ms được thực hiện giống hệt nhau (Giống với chép phạt các nội dung được lặp đi lặp lại), vì thế sử dụng cấu trúc lệnh lặp xác định để thiết kế lại chương trình và ta có chương trình như sau:



Hình 7.4: Chương trình tạo âm báo thức sử dụng lặp

Tiến hành thử nghiệm chương trình trên Board và lắng nghe âm thanh được tạo ra tương ứng với chương trình này. Kết quả cho thấy nhạc chuông được tạo ra vẫn đảm bảo như chương trình phía trên và chương trình đã được rút gọn hơn chương trình ban đầu rất nhiều.

Chương trình được phân tích chi tiết như sau:

Ban đầu chương trình tạm dừng 1000ms(1s) sau đó thực hiện 1 vòng lặp 4 lần, bên trong vòng lặp đó cụm lệnh thực thi (Nằm trong viền màu đỏ), tức là cụm lệnh thực thi này sẽ được

chương trình cho phép chạy liên tục 4 lần. Sau khi hết 4 lần liên tiếp bật và tắt âm thanh trên còi chương trình tự động quay lại lệnh đầu tiên (tạm dừng 1000ms) để tạo thành các nhịp âm báo thức liên tục.

Ví dụ trên cho thấy với việc sử dụng vòng lặp chương trình đã được thu gọn rất nhiều. Vì vậy, vòng lặp xách định nói riêng và vòng lặp nói chung giúp chúng ta xây dựng chương trình ngắn gọn hơn và dễ dàng hình dung về nội dung chương trình hơn.

2. Lệnh lặp xác định với biến

Ví dụ 2: Xây dựng chương trình hiển thị các số từ 0-10 lên màn hình hiển thị số. Mỗi số được hiển thị lên màn hiển thị trong 1 giây.

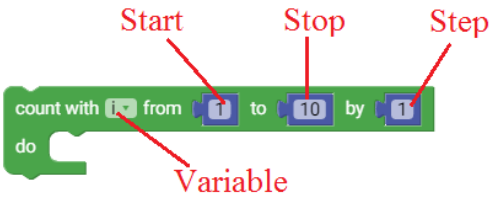
Giải pháp đơn giản nhất để thực hiện bài toán là thiết kế một chương trình sử dụng liên tiếp các hàm hiển thị số và đưa vào đó các số từ 0 đến 10. Giữa các lần hiển thị thực hiện tạm dừng chương trình 1000ms. Chương trình đó có nội dung như hình dưới đây:

Hình 7.5: Chương trình hiển thị số từ 0-10 không sử dụng lặp

Với độ dài như thế này của chương trình, thử tưởng tượng nếu chương trình yêu cầu hiển thị số đếm từ 0-9999 thì có lẽ chúng ta không đủ kiên trì để xây dựng một chương trình theo cách như trên.

Và giải pháp ta nghĩ đến là sử dụng vòng lặp. Ở ví dụ trên, ta thấy nội dung các lệnh được thực thi trong vòng lặp là giống nhau hoàn toàn vậy đối với trường hợp này thì sao? Ở chương trình này ta thấy rằng về cấu trúc các lệnh là tương đương nhau tuy nhiên có sự khác biệt về

nội dung số cần để hiển thị lên màn hình hiển thị số. Các số được hiển thị ở đây có liên quan đến số vòng lặp vì vậy chúng ta vẫn có thể sử dụng vòng lặp để tiến hành xây dựng chương trình này thông qua lệnh lặp với biến.



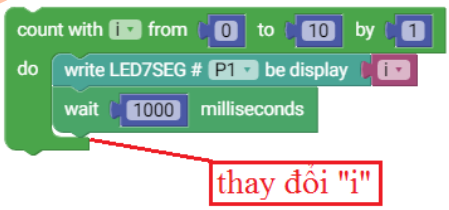
Hình 7.6: Cấu trúc lệnh lặp với biến

Lệnh lặp với biến ngoài cho phép thực hiện tập lệnh thực thi bên trong với số lần xác định mà còn tạo cho một biến có giá trị thay đổi sau mỗi bước lặp. Cụ thể như sau:

Tên mục	Chi tiết
Variable (Tên biến)	Biến dùng để đếm vòng lặp. Giá trị của biến phụ thuộc vào các giá trị: Start, Stop, Step. Các lệnh thực thi bên trong lệnh lặp có thể sử dụng biến này như một biến thông thường.
Start (Giá trị ban đầu)	Giá trị ban đầu được truyền cho Variable khi bắt đầu vòng lặp
Stop (Giá trị ngưỡng)	Giá trị ngưỡng để kết thúc vòng lặp: Ví dụ 1: Nếu Start = 5, Stop = 10, Start<Stop: vòng lặp được kết thúc khi giá trị của Variable lớn hơn 10(Stop) Ví dụ 2: nếu Start = 10, Stop = 5, Start>Stop: vòng lặp được kết thúc khi giá trị của Variable nhỏ hơn 5(Stop)
Step (Bước nhảy)	Giá trị thay đổi của Variable sau mỗi lần các lệnh lặp được thực thi. Ví dụ: Start = 0, Stop = 11, Step = 2. Thì Variable lần lượt có các giá trị như sau: 0,2,4,6,8,10, (Không có 12 vì 12>11(Stop))

Bảng 7.2: Các thông số của lệnh lặp với biến

Quay lại với bài toán ở trên, ta có thể sử dụng vòng lặp với biến để giải quyết bài toán trên bằng cách thay vì đưa trực tiếp số cần hiển thị vào hàm hiển thị số, ta truyền số cần hiển thị vào một biến và cung cấp biến đó cho hàm hiển thị. Chương trình cụ thể như sau:



Hình 7.7: Chương trình hiển thị số từ 0-10 sử dụng lặp với biến

Phân tích về chương trình:

Bước 1: Ban đầu biến “i” được khởi tạo với giá trị ban đầu là 0.

Bước 2: Cụm lệnh bên trong vòng lặp được thực thi: Biến "i" được cập nhật vào hàm hiển thị và đưa ra màn hình trong 1 giây.

Bước 3: Tại vùng màu đỏ (Cuối cùng của vùng lệnh lặp) biến "i" tự động được thay đổi theo Step quy định.

Bước 4: Quay lại kiểm tra nếu biến i vẫn nằm trong vùng cho phép thì quay lại bước 2. Nếu đã vượt vùng cho phép thì kết thúc vòng lặp.

Như vậy ban đầu i có giá trị là 0 và giá trị của i được đưa ra hiển thị lên màn hình hiển thị số trong 1 giây. Sau đó, biến i lần lượt được tăng 1 đơn vị được hiển thị ra màn hình. Khi i đạt giá trị 10 và đã được hiển thị lên màn hình đủ 1s biến i được tăng lên 1 đơn vị và trở thành 11. 11 là số đã vượt vùng cho phép (11>10) nên vòng lặp này được kết thúc. Như vậy, chương trình này đảm bảo giúp chúng ta hiển thị các con số từ 0-10 một cách lần lượt với kết cấu ngắn gọn hơn nhiều.

Ví dụ 3: Hiển thị các số chẵn trong khoảng từ 100 đến 1000, mỗi số được hiển thị trong 1 giây.

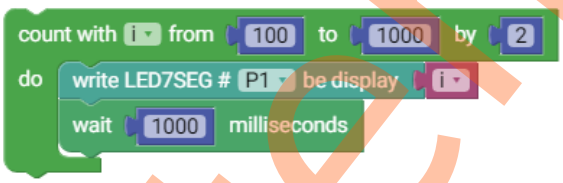
Phân tích: Các số chẵn là các số chia hết cho 2 hay các số có kết thúc là: 0,2,4,6,8.

Vì số chẵn cách đều nhau là 2 đơn vị nên ta sử dụng 2 làm Step.

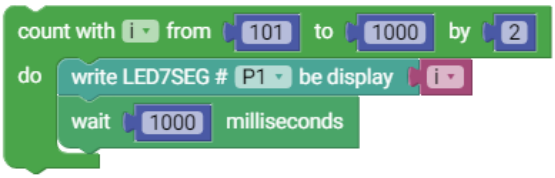
Ta có 100 là số chẵn và là số chẵn nhỏ nhất trong khoảng từ 100-1000. Chọn giá trị 100 làm giá trị Start. Nếu bài toán yêu cầu hiển thị số lẻ thì ta phải chọn số lẻ nhỏ nhất làm Start. Vì vậy nếu bài toán yêu cầu hiển thị số lẻ thì giá trị Start là 101.

Giá trị Stop là giá trị lớn nhất mà biến có thể sử dụng vì vậy ở đây Stop bằng 1000.

Sau khi đã phân tích xong ta thiết kế được chương trình như sau:



Hiển thị số chẵn



Hiển thị số lẻ

Hình 7.8: Chương trình hiển thị số chẵn và số lẻ

Như vậy, bằng cách sử dụng vòng lặp và thay đổi các thông số Start, Stop, Step đã giúp chúng ta giải quyết được nhiều bài toán tưởng chừng như phức tạp trở thành những chương trình đơn giản và gọn gàng hơn rất nhiều.

3. Lệnh điều khiển lặp

Ví dụ 4: Xây dựng chương trình hiển thị các số nguyên từ 0-100 trừ số 49 và 53 lên màn hình hiển thị số, mỗi số được hiển thị 1 giây.

Trong quan niệm của người Việt Nam chúng ta thì 2 con số 49 và 53 là 2 con số xấu nên mọi người thường muốn tránh. Để phục vụ bài học chúng ta xây dựng một chương trình hiển thị số từ 0 đến 100 nhưng tránh xuất hiện 2 con số này dựa vào một số phương pháp như sau:

Cách 1: Cách đơn giản có thể thực hiện là sử dụng kiến thức của bài học số 4. Ta thực hiện kiểm tra giá trị của mỗi biến trước khi đưa vào hàm hiển thị, nếu giá trị của biến không phải là số 49 hoặc 53 thì cho phép hiển thị ngược lại không cho phép hiển thị. Vậy chương trình được thiết kế được cấu trúc như sau:

Hình 7.9: Điều khiển chương trình lặp sử dụng điều kiện nếu-thì

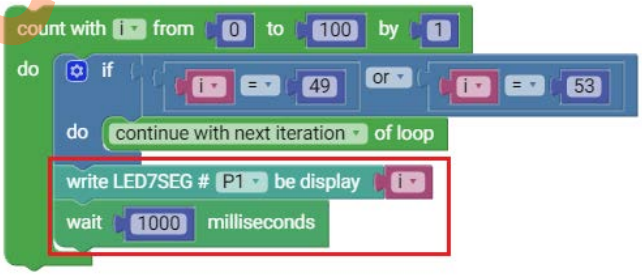
Phân tích chương trình: Biến i được thay đổi giá trị trong khoảng từ 0 đến 100 với mỗi lần thay đổi tăng lên 1 đơn vị (Step = 1). Trong lệnh thực thực thi của lệnh lặp thực hiện kiểm tra nếu giá trị của biến i không phải là số 49 và 53 (i khác 49 và khác cả 53) thì thực hiện hiển thị số lên màn hình. Vì thế nếu i có giá trị bằng 49 hoặc 53 thì chương trình không thực hiện hiển thị mà ngay lập tức tăng biến i để thực hiện một vòng lặp mới.

Cách 2: Cách 2 chúng ta khám phá một lệnh điều khiển vòng lặp. Lệnh điều khiển này là lệnh "**continue**".

Hình 7.10: Lệnh "continue" trong điều khiển chương trình lặp

Lệnh "**continue**" cho phép thực hiện một vòng lặp mới không cần phải thực hiện toàn bộ các lệnh nằm bên trong khối lệnh thực thi của lệnh lặp đang thực thi. Khi chương trình gặp lệnh này ngay lập tức biến đếm của vòng lặp sẽ được thay đổi theo giá trị theo Step và bỏ qua tất cả các lệnh phía sau (nếu có) bên trong khối lệnh thực thi của lệnh lặp.

Với ví dụ này chương trình được xây dựng bằng lệnh "**continue**" như sau:



Hình 7.11: Chương trình sử dụng lệnh "continue" trong điều khiển lặp

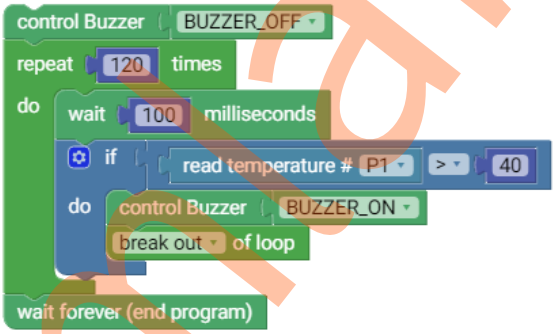
Ta thấy rằng ở đây có lệnh kiểm tra nếu biến i có giá trị bằng 49 hoặc 53 thì thực hiện lệnh "**continue**. Khi lệnh "**continue**" đực thực hiện các lệnh phía sau (gồm 2 lệnh trong vùng khoanh đỏ) được bỏ qua và biến i được thay đổi theo Step để thực hiện một chu kỳ lặp mới.

Ví dụ 5: Giả sử quá trình kiểm tra hoạt động của quá trình khởi động một động cơ đang được nghiên cứu sản xuất có một tiêu chí như sau:

- Quá trình khởi động động cơ kéo dài 2 phút (120s).
- Nếu trong 120s này nếu nhiệt độ vượt ngưỡng 40°C thì chứng tỏ động cơ chưa đạt yêu cầu, thiết bị bật còi cảnh báo để nhắc nhở nhân viên thử nghiệm. (Thực tế nhiệt độ động cơ không thấp như thế nhưng để đảm bảo điều kiện thử nghiệm phù hợp với cảm biến ta lấy con số này để có thể dễ dàng có thể tăng giảm nhiệt độ).

Phân tích: Quá trình thử nghiệm diễn ra tối đa trong 120s với điều kiện trong 120s này nhiệt độ của động cơ không được cao hơn 40°C. Nếu trong 120s này chỉ có một lần 40°C nhiệt độ cao hơn 40°C chứng tỏ quy trình thử nghiệm này không đạt và có thể không cần phải thử nghiệm khoảng thời gian còn lại của quá trình thử nghiệm.

Dựa vào phân tích trên chúng ta thiết kế được chương trình như sau:



Hình 7.12: Chương trình kiểm tra quá trình khởi động động cơ

Với ví dụ này ta tìm hiểu về lệnh "**break out**". Lệnh "**break out**" là lệnh điều khiển vòng lặp. Khác với "**continue**" cho phép chương trình bỏ qua một số lần lặp nào đó, "**break out**" cho phép chương trình thoát khỏi hoàn toàn lệnh lặp đang thực hiện.

Chú ý: "**break out**" cho phép chương trình thoát khỏi lệnh lặp nằm gần nó nhất từ bên trái sang mà lệnh "**break out**" nằm trong.

Với ví dụ trên, chương trình thực hiện một vòng lặp 120 lần và mỗi lần cách nhau 1 giây để đảm bảo quá trình kiểm nghiệm được diễn ra trong 120 giây. Trong 120 lần này chương trình thực hiện đọc nhiệt độ và so sánh với 40°C. Nếu nhiệt độ không lớn hơn 40°C thì lệnh bật còi cảnh báo và "**break out**" không được thực hiện. Vì vậy, nếu sau 120 lần không có lần nào nhiệt độ lớn hơn 40°C thì lệnh bật còi cảnh báo không được thực thi. Vì lệnh bật còi không được thực thi nên không có tiếng còi kêu trong và sau quá trình thử nghiệm. Tuy nhiên, nếu trong 120 lần kiểm tra này chỉ cần xuất hiện 1 lần nhiệt độ cao hơn 40°C thì lệnh bật còi cảnh báo và lệnh "**break out**" được thực hiện. Lệnh "**break out**" thực hiện thoát chương trình

khỏi vòng lặp gần nhất mà nó nằm trong. Vì thế khi lệnh lặp kết thúc thì chương trình sẽ thực thi lệnh "**wait forever**", và do lệnh bật còi đã được thực hiện nên còi sẽ tiếp tục kêu cho đến khi có sự tác động đến thiết bị từ người sử dụng.

Như vậy, với ví dụ này chúng ta đã tìm hiểu thêm được cách điều khiển các lệnh lặp bao gồm **"continue"** và **"break out"** trong trường hợp cần thiết. Đây là một trong những lệnh điều khiển có thể thường gặp khi xử lý các chương trình liên quan đến lặp vì vậy chúng ta cần thực hành nhiều hơn để nắm rõ kiến thức về phần này.

BÀI 8: LỆNH LẶP KHÔNG XÁC ĐỊNH

Trên đây là hình ảnh ví dụ về lặp không xác định. Giả sử ta biết trước ở cuối đường hầm có một kho báu và chưa biết khoảng cách từ chỗ ta đứng tới vị trí kho báu là bao xa. Công việc của chúng ta là tiến hành đào đất để mở đường tiến về hướng kho báu. Hình trên có hai người đại diện cho 2 cách tiếp cận khác nhau:

1. Người mặc áo trắng bên dưới: Đào một khoảng hầm xác định theo số lần đào xác định trước nào đó.
2. Người mặc áo xanh ở bên trên: Đào liên tục cho đến khi nào tìm được kho báu thì dừng lại.

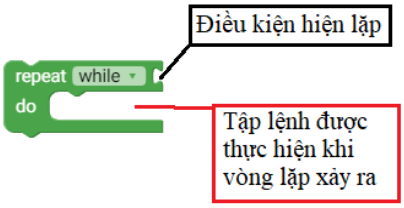
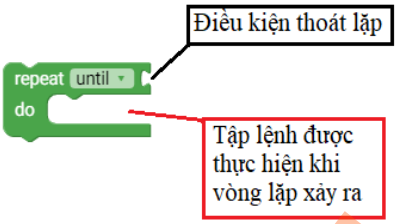
Trường hợp số 1: Người đàn ông áo trắng sau khi thực hiện một số lần đào nhất định đã xác định trước nhưng không thấy kho báu xuất hiện nên ông ta đã dừng công việc mặc dù ông ấy đã ở rất gần với kho báu. Có lẽ chỉ cần thêm một số lần đào nữa thì ông ấy sẽ tìm thấy cả một kho báu lớn dành cho mình. Tuy nhiên, do đã đặt hạn mức số lần đào nên khi đạt đủ số lần đó ông ấy dừng lại và kết quả là không tìm thấy kho báu.

Trường hợp số 2: Hiện tại, anh ta mới đào được một đoạn ngắn hơn nhiều so với người đàn ông ở dưới nhưng anh ta vẫn đang tiếp tục đào. Mục tiêu của anh ta là sẽ đào cho đến khi tìm thấy kho báu thì mới dừng lại. Và nếu như anh ta vẫn đặt mục tiêu đào cho đến khi thấy kho báu thì chắc chắn anh ta sẽ tìm thấy. Có thể mô tả các thực thi của trường hợp này như sau: Tiếp tục thực hiện đào cho đến khi kho báu xuất hiện.

Trên đây chính là ví dụ về 2 kiểu lặp thường gặp trong lập trình. Kiểu lặp dành cho người đàn ông áo trắng phía dưới là kiểu lặp xác định đã giới thiệu ở bài trước. Kiểu lặp không xác định chính là kiểu lặp lại việc đào đất của người đàn ông áo vàng phía trên. Sự sai khác nhau chính của 2 kiểu lặp này là: Một bên xác định trước số công việc sẽ làm còn một bên thực hiện công việc không rõ khối lượng chỉ biết được công việc sẽ được dừng lại khi đã đạt được một điều kiện nào đó đặt ra từ trước.

Ví dụ khác: Khi may quần áo, thợ may không xác định trước sẽ khâu bao nhiêu đường kim mũi chỉ mà mục tiêu của người thợ may là thực hiện việc khâu may cho đến khi chiếc áo được hoàn thiện và có thể mặc được. Tùy thuộc vào kiểu quần áo, chất liệu sử dụng mà số lượng đường may dành cho mỗi sản phẩm là khác nhau nhưng chúng ta vẫn có thể tiến hành một vòng lặp cho công việc này với điều kiện mục tiêu đã xác định từ trước.

Cấu trúc lệnh lặp không xác định trong CloverBlock có 2 dạng chính như sau:

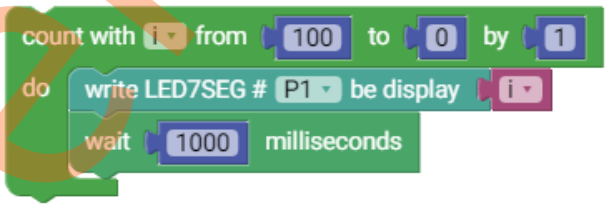
STT	Block	Vị trí	Chức năng
1		Loop	Thực hiện khối lệnh thực thi khi điều kiện lặp vẫn đúng. Khi điều kiện lặp không còn đúng nữa thì vòng lặp dừng lại.
2		Loop	Thực hiện khối lệnh thực thi cho đến khi điều kiện kiểm tra đúng, nếu điều kiện kiểm tra vẫn chưa đúng thì vẫn tiếp tục thực thi các lệnh bên trong vòng lặp.

Bảng 8.1: Các lệnh lặp không xác định

Chú ý: Hai lệnh này có thể coi là tương đương và có thể sử dụng thay thế cho nhau nếu cần thiết. (Tham khảo lệnh **NOT** nằm trong mục Logic)

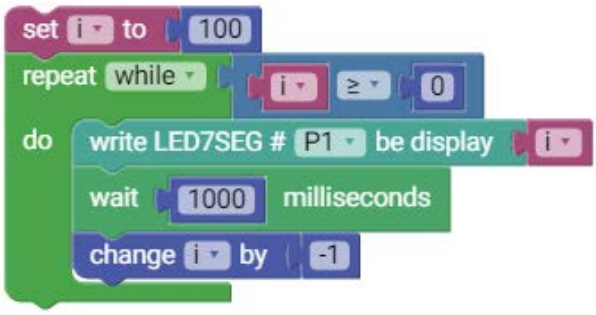
Ví dụ 1: Thực hiện hiển thị lần lượt giảm dần các số không âm xuất phát từ 100.

Phân tích: Quay lại với bài toán hiển thị số lên màn hình hiển thị số. Bài toán này có thể sử dụng vòng lặp xác định thực hiện lặp từ 100 về đến 0 với Step là 1 như sau:



Hình 8.13: Chương trình hiển thị số giảm dần sử dụng lặp xác định

Trên đây là một cách để thực hiện, nhưng bài toán này cũng có thể được thực hiện bởi vòng lặp không xác định bằng cách đặt giá trị ban đầu cho biến số hiển thị, sau mỗi lần hiển thị chúng ta thực hiện giảm biến đi 1 đơn vị (chiều giảm dần) và nếu số đó còn lớn hơn hoặc bằng 0 (Không âm) thì tiếp tục thực hiện việc công việc lặp hiển thị và giảm số. Với logic đó ta thiết kế được một chương trình như sau:



Hình 8.14: Chương trình hiển thị số giảm dần sử dụng lệnh lặp không xác định

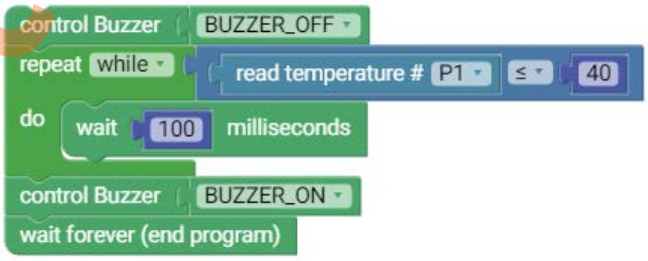
Để kiểm tra tính đúng đắn của chương trình, tiến hành nạp chương trình vào Board và tiến hành quan sát để đưa ra đánh giá. Nếu tiến trình diễn ra quá chậm, có thể thay đổi giá trị lớn nhất 100 về giá trị nhỏ hơn hoặc giảm thời gian của lệnh chờ giữa các lần thay đổi giá trị để tốc độ của tiến trình diễn ra được nhanh hơn.

Ví dụ 2: Thiết kế hệ thống cảnh báo an toàn của buồng ủ bảo quản hóa chất hoạt động theo nguyên tắc như sau:

- Buồng ủ hoạt động tốt khi nhiệt độ luôn đảm bảo có nhiệt độ nhỏ hơn 40°C .
- Nếu thiết bị điều khiển nhiệt độ không tốt nhiệt độ có thể cao hơn 40°C chứng tỏ thiết bị điều khiển nhiệt độ có hỏng hóc.
- Khi nhiệt độ cao hơn 40°C thì hóa chất bị hỏng, ngay lập tức đưa ra cảnh báo bằng còi kêu.

Phân tích: Đây là một thiết bị hoạt động rất nghiêm ngặt. Nếu như các bài trước về cảnh báo cháy rừng thì các mức độ có thể thay đổi nhưng với thiết bị này khi nhiệt độ chỉ cần cao hơn 40°C trong một thời gian rất ngắn cũng đồng nghĩa với hóa chất được bảo quản đã hỏng và không thể sử dụng được nữa. Vì vậy, dù sau đó nhiệt độ có xuống thấp hơn 40°C thì hóa chất này cũng không còn giá trị sử dụng vì vậy còi cảnh báo vẫn phải tiếp tục kêu để báo cho người giám sát biết rằng điều kiện bảo quản đã bị vi phạm.

Thiết kế: Thiết bị cảnh báo (Còi) vẫn được tắt cho đến khi gặp điều kiện nhiệt độ cao hơn 40°C. Khi nhiệt độ đọc được cao hơn 40°C tiến hành bật còi cảnh báo và dừng toàn bộ tiến trình đọc nhiệt độ vì lúc này việc đọc nhiệt độ đã trở thành không cần thiết (Hóa chất đã hỏng). Sau khi phân tích chương trình được xây dựng như sau:

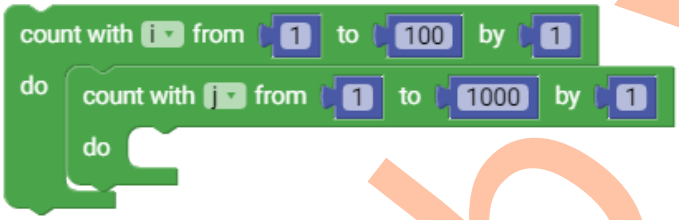


Hình 8.15: Chương trình giám sát nhiệt độ bảo quản hóa chất

- Khi thiết bị khởi động lên còi cảnh báo được tắt.

- Thiết bị thực hiện đọc nhiệt độ và so sánh với 40.
- Nếu nhiệt độ còn thấp hơn 40°C chứng tỏ lô hóa chất vẫn còn an toàn thiết bị cần tiếp tục đọc nhiệt độ để kiểm tra. Các lần đo đạc cách nhau 100ms.
- Khi điều kiện nhiệt độ bị vi phạm (Nhiệt độ lớn hơn 40°C) hàm kiểm tra cho giá trị là False thì vòng lặp dừng lại và lệnh bật còi cảnh báo được thực hiện ngay sau vòng lặp và chương trình dừng lại để cảnh báo được duy trì cho đến khi người giám sát tác động và khởi động lại thiết bị giám sát.

Chú ý: Các lệnh lặp có thể thực hiện lồng ghép nhau nhiều lần cho phép tạo ra chương trình sử dụng lệnh lặp lồng. Cấu trúc lệnh lặp lồng có thể thấy dạng như sau:



Hình 8.16: Ví dụ về cấu trúc lệnh lặp lồng nhau

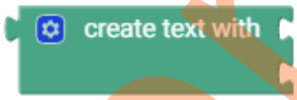
Với cấu trúc lệnh lặp lồng, lệnh lặp bên trong sẽ được thực hiện với số lần được thực hiện bằng với số lần lặp trong 1 lần của lệnh lặp phía ngoài.

BÀI 9: CHUỖI VĂN BẢN (STRING)

Chuỗi văn bản (String) là một dãy gồm các ký tự. Với CloverBlock, chuỗi văn bản là cách thức chính giúp chúng ta thực hiện giao tiếp giữa Board với máy tính thông qua kết nối USB hoặc với điện thoại thông minh thông qua kết nối không dây Bluetooth/Wifi. Khi chúng ta tiến hành gửi đi một lệnh điều khiển từ máy tính hoặc điện thoại xuống thiết bị thường chúng ta có xu hướng gửi lệnh gần với ngôn ngữ tự nhiên chúng ta sử dụng như: BAT DEN, TAT DEN, TIEN LEN, LUI LAI,... hoặc khi chúng ta gửi từ Board lên máy tính hoặc điện thoại một thông số nào đó chúng ta thường đính kèm thêm một đoạn ký tự để người quan sát biết được thông số đang truyền lên là thông số gì. Đây chính là các ví dụ mà chúng ta sử dụng chuỗi văn bản trong khi lập trình với CloverBlock nói riêng và lập trình nói chung. Đối với lập trình nói chung chuỗi văn bản còn được sử dụng vào nhiều ứng dụng và nhiệm vụ khác.

Phần này giới thiệu một số phương pháp làm việc với chuỗi văn bản để tạo ra các chuỗi văn bản theo ý muốn, gửi dữ liệu từ máy Board lên máy tính thông qua kết nối USB hoặc điện thoại thông qua Bluetooth và nhận dữ liệu từ máy tính/điện thoại sử dụng CloverBoard. Ngoài ra, nâng cao hơn nữa chúng ta sẽ tìm cách sử dụng các lệnh được gửi từ máy tính và điện thoại đến Board để điều khiển Board hoạt động theo ý muốn.

Dưới đây là chi tiết về một số lệnh được sử dụng với chuỗi văn bản (String) mà CloverBlock hỗ trợ:

STT	Block	Vị trí	Chức năng
1		Text	Tạo mỗi chuỗi văn bản cố định.
2		Text	Ghép nhiều chuỗi văn bản, số… thành một chuỗi văn bản mới.
3		Text	Thêm một chuỗi văn bản vào phía sau một biến kiểu văn bản.
4		Text	Đếm số ký tự của một chuỗi văn bản.
5		Text	Kiểm tra một chuỗi văn bản có phải là chuỗi văn bản trống không (Không có ký tự nào cả)
6		Text	Kiểm tra một chuỗi văn bản nào đó có nằm trong một chuỗi văn bản khác không. Ví dụ: “Hello” nằm trong “Hello Clover”

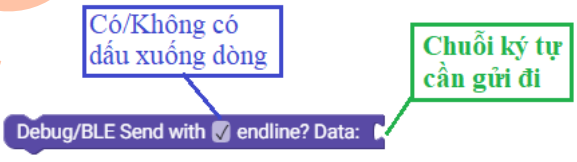
STT	Block	Vị trí	Chức năng
7		Debug/ Bluetooth	Biến dùng để lưu dữ liệu nhận được từ máy tính hoặc điện thoại
8		Debug/ Bluetooth	Kiểm tra chuỗi văn bản nhận được từ máy tính/điện thoại có chứa một chuỗi văn bản nào đó không (Tương tự 6)
9		Debug/ Bluetooth	Kiểm tra có dữ liệu vừa được gửi đến từ máy tính hoặc điện thoại hay không
10		Debug/ Bluetooth	Thực hiện sao chép dữ liệu nhận được từ **điện thoại** vào biến ở mục số 7. Nên thực hiện ngay khi phát hiện có dữ liệu mới được gửi đến.
11		Debug/ Bluetooth	Thực hiện sao chép dữ liệu nhận được từ **máy tính** vào biến ở mục số 7. Nên thực hiện ngay khi phát hiện có dữ liệu mới được gửi đến.
12		Debug/ Bluetooth	Gửi một chuỗi văn bản từ Board đến máy tính hoặc điện thoại.

Bảng 9.1: Các lệnh liên quan đến chuỗi văn bản của CloverBlock

1. Gửi dữ liệu từ Board lên máy tính và điện thoại

Với việc sử dụng CloverBoard để lập trình, thiết kế các ý tưởng. Trong quá trình thiết kế có thể ta cần tiến hành gửi dữ liệu lên máy tính hoặc điện thoại để dễ dàng quan sát hoạt động của chương trình ở nhiều chế độ, điều kiện khác nhau nhằm đảo bảo chương trình đã hoạt động đúng theo yêu đặt ra. Ngoài ra, việc gửi dữ liệu lên máy tính hoặc điện thoại còn giúp chúng ta quan sát dữ liệu đọc được từ một số cảm biến trong trường hợp chúng ta không muốn hiển thị lên màn hình mà vẫn muốn quan sát dữ liệu.

Để có thể gửi dữ liệu lên máy tính hoặc điện thoại ta có thể sử dụng hàm số 12 ở bảng trên, chi tiết như sau:



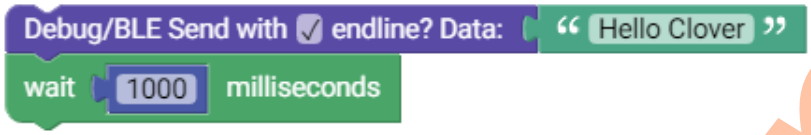
Hình 9.1: Lệnh gửi dữ liệu lên điện thoại/máy tính

Chuỗi ký tự cần gửi đi có thể là một chuỗi cố định hoặc một biến có kiểu dữ liệu là chuỗi văn bản.

Dấu tích lựa chọn sử dụng để chọn có hoặc không có dấu xuống dòng ở cuối của chuỗi văn bản gửi đi. Dấu xuống dòng cho phép chúng ta nhìn dữ liệu mỗi lần gửi đi được gửi đi hiển thị trong một dòng độc lập vì thế sẽ dễ dàng cho chúng ta quan sát và phân tích.

Ví dụ 1: Gửi dòng chữ "Hello Clover" từ Board lên máy tính và điện thoại theo chu kỳ 1 giây.

Bằng việc sử dụng hàm gửi dữ liệu từ Board Clover lên ta có chương trình sau:

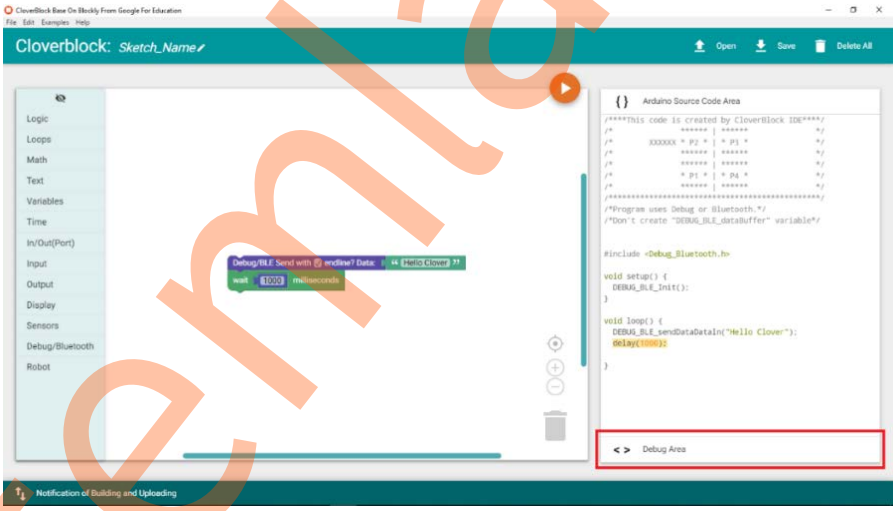


Hình 9.2: Chương trình gửi "Hello Clover" lên máy tính/điện thoại

Chương trình sử dụng hàm gửi dữ liệu với đối tượng truyền vào là một chuỗi văn bản cố định là "Hello Clover". Sau mỗi lần gửi chương trình sẽ tạm dừng 1000ms (1 giây) và tiếp tục gửi dữ liệu lên máy tính hoặc điện thoại.

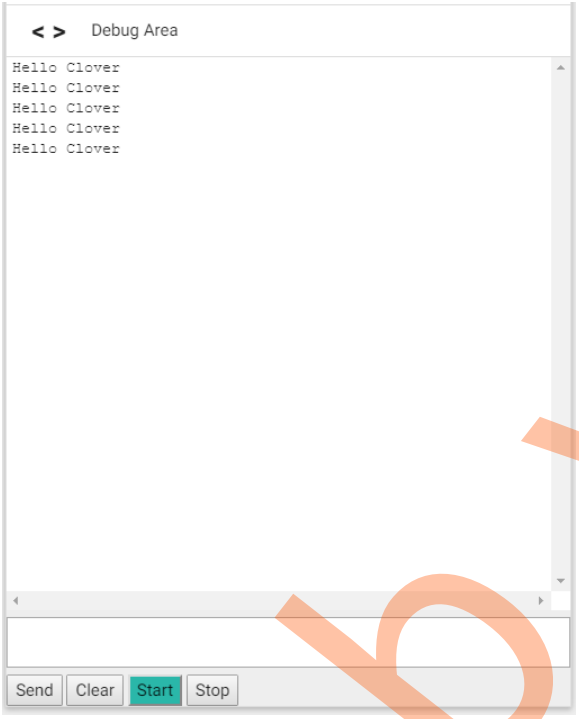
Thử nghiệm với máy tính: Để thử nghiệm quá trình truyền dữ liệu lên máy tính đã thành công hay chưa chúng ta thực hiện các bước sau:

- Bước 1: Chuyển chế độ của nút Run/Boot trên Board về chế độ **Boot** để kết nối Board với máy tính (Nút nhấn ở mức thấp).
- Bước 2: Mở vùng **Debug Area** trong CloverBlock bằng cách nhấn vào chữ Debug Area ở bên phải của giao diện phần mềm.



Hình 9.3: Mở giao diện kết nối dữ liệu với Board trên máy tính

- Bước 3: Sau khi giao diện Debug Area đã hiện ra. Bấm vào nút "**Start**" để bắt đầu quan sát dữ liệu truyền từ Board lên máy tính. Trong quá trình quan sát dữ liệu chúng ta có thể nhấn nút "**Clear**" để xóa toàn bộ dữ liệu nhận được từ Board hoặc nút "**Stop**" để dừng quá trình quan sát dữ liệu nhận từ Board.
  Dữ liệu quan sát được từ chương trình trên như sau:

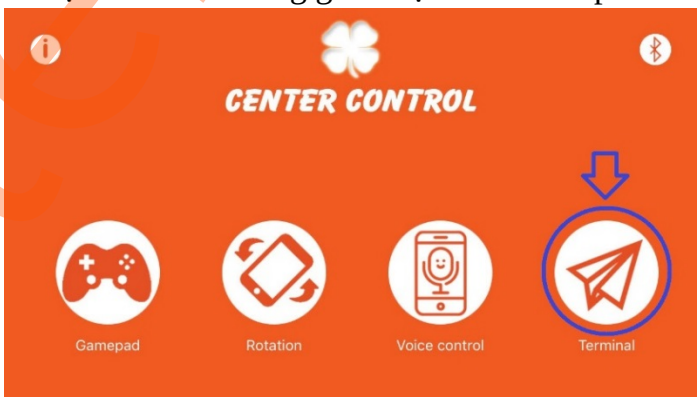


Hình 9.4: Kết quả thử nghiệm chương trình gửi liệu từ Board lên máy tính

Như vậy chúng ta đã kiểm nghiệm được quá trình truyền dữ liệu từ Board lên máy tính thành công.

Thử nghiệm với điện thoại: Để có thể thử nghiệm quá trình truyền nhận dữ liệu giữa điện thoại và Board chúng ta cần thực hiện các bước sau:

- Bước 1: Kết nối module Bluetooth của bộ thiết bị vào vị trí cần kết nối sau đó chuyển chế độ của nút Run/Boot trên Board về chế độ **Run** để kết nối Board với điện thoại (Nút nhấn ở mức cao).
- Bước 2: Khởi động phần mềm Clover Master và sử dụng phần mềm để thực hiện kết nối giữa điện thoại với Bluetooth của Board.
- Bước 3: Chọn chế độ **Terminal** trong giao diện chính của phần mềm Clover Master.



Hình 9.5: Mở giao diện truyền nhận dữ liệu trên điện thoại

- Bước 4: Quan sát dữ liệu mà Board gửi đến điện thoại tại giao diện này. Có thể sử dụng nút "**Clear**" để xóa dữ liệu quan sát cũ nếu cần thiết.

Hình 9.6: Kết quả thử nghiệm truyền dữ liệu từ Board lên điện thoại

Chương trình gửi dữ liệu đến máy tính và điện thoại là tương đương vì vậy các thử nghiệm từ bài viết sẽ này sẽ chỉ dừng lại ở thử nghiệm trên máy tính. Người đọc có thể thử nghiệm lại trên điện thoại nếu cần thiết.

Ví dụ 2: Gửi dữ liệu nhiệt độ đọc được từ Board lên máy tính hoặc điện thoại.

Phân tích: Ta thấy nhiệt độ đọc được có dữ liệu trả về là dạng số, trong khi yêu cầu của hàm gửi dữ liệu là dữ liệu cần gửi đi cần phải có kiểu dữ liệu là dạng chuỗi văn bản (String). Vì vậy, để có thể sử dụng hàm gửi dữ liệu từ Board lên máy tính hoặc điện thoại cần phải sử dụng hàm chuyển đổi kiểu dữ liệu *(Đã được nhắc đến ở phần biến)* để chương trình có thể hoạt động đúng yêu cầu. Chương trình cụ thể như sau:

Hình 9.7: Chương trình gửi dữ liệu nhiệt độ lên máy tính

Kết quả được hiển thị trên Debug Area như sau:

Hình 9.8: Kết quả thử nghiệm gửi nhiệt độ lên máy tính

Với chương trình này, thông số gửi lên chỉ là số thông thường, trong trường hợp ta muốn chương trình gửi lên thông số của nhiều cảm biến, nhiều thông số khác hoặc người khác muốn quan sát thì sẽ không thể hiểu được thông số này là thông số gì. Vì vậy, ta nên sử dụng *(Không bắt buộc)* thêm một đoạn văn bản phía trước để xác định rõ dữ liệu gửi đi là

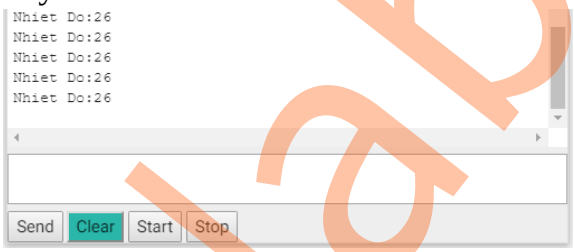
thông số gì. Ở đây ta đo nhiệt độ môi trường nên ta thêm cụm từ "Nhiet Do:" phía trước để tiện quan sát.

Có 2 cách để giải quyết như sau:

Cách 1: Lần lượt gửi đoạn văn bản rồi gửi thông số nhiệt độ. Ta có thể sử dụng hai hàm gửi dữ liệu liên tiếp nhau với lệnh gửi dữ liệu đầu tiên gửi đi chuỗi "Nhiet Do:" và lệnh gửi thứ hai là gửi đi thông số nhiệt độ như chương trình trên đã hoàn thành. Để đảm bảo 2 dữ liệu này cùng 1 dòng ta chỉ chọn gửi kèm dấu xuống dòng ở lệnh gửi thông số nhiệt độ còn lệnh gửi chuỗi văn bản không nên gửi kèm lệnh xuống dòng. Chương trình cụ thể như sau:

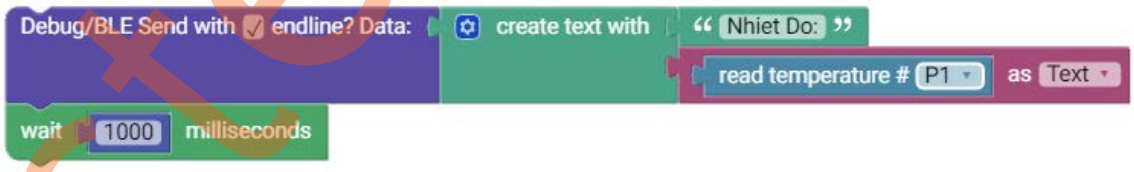
Hình 9.9: Chương trình gửi dữ liệu nhiệt độ kèm thông tin lên máy tính (1)

Kết quả hiển thị trên máy tính:



Hình 9.10: Kết quả thử nghiệm gửi dữ liệu nhiệt độ kèm thông tin lên máy tính (1)

Cách 2: Ghép 2 chuỗi văn bản "Nhiet Do:" và thông số nhiệt độ rồi tiến hành gửi đi một lần. Ở phần trên đã giới thiệu hàm ghép các chuỗi văn bản làm một chuỗi. Vì thế, ta có thể sử dụng cách này để gửi dữ liệu đi. Với cách giải quyết này ta có thể dùng biến để lưu dữ liệu đoạn văn bản được ghép sau đó đưa biến đó vào hàm gửi dữ liệu đi hoặc sử dụng trực tiếp của lệnh ghép văn bản và tiến hành gửi đi. Hai cách làm trên đều cho kết quả tương đương. Hai chương trình đó cụ thể như sau:

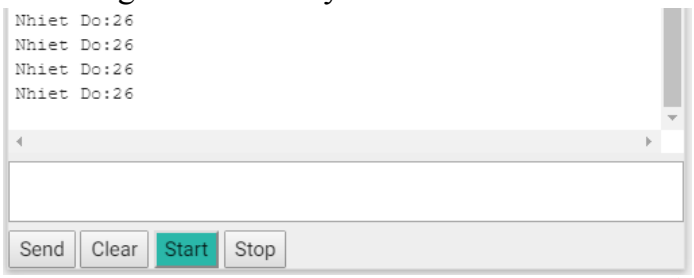


Chương trình không sử dụng biến

Chương trình sử dụng biến đệm

Hình 9.11: Chương trình gửi dữ liệu nhiệt độ kèm thông tin lên máy tính (2)

Kết quả hiển thị của chương trình trên máy tính như sau:



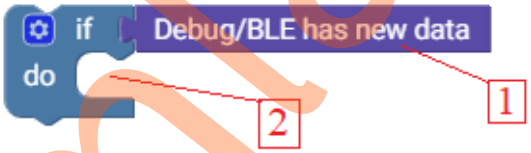
Hình 9.12: Kết quả thử nghiệm gửi dữ liệu nhiệt độ kèm thông tin lên máy tính (2)

Như vậy, hai cách giải quyết trên có chung một kết quả hiển thị đầu ra. Vì vậy, trong quá trình sử dụng tùy thuộc vào thói quen chúng ta chọn cách phù hợp để sử dụng.

2. Nhận dữ liệu từ máy tính điện thoại.

Ở phần trước chúng ta đã tìm hiểu cách gửi dữ liệu từ điện thoại lên máy tính và điện thoại theo các cách khác nhau. Phần này giới thiệu cách nhận dữ liệu nhận được từ điện thoại, máy tính từ đó làm nền tảng thực hiện điều khiển thiết bị sử dụng lệnh điều khiển được gửi đi từ máy tính thông qua kết nối USB hoặc từ điện thoại thông qua Bluetooth.

Mỗi khi có dữ liệu mới được gửi từ máy tính hoặc điện thoại đến sẽ có một thông báo với chương trình rằng có dữ liệu mới đến. Vì vậy, trong chương trình cần có một đoạn lệnh liên tiếp kiểm tra có xuất hiện thông báo này hay không. Cụ thể cấu trúc một chương trình nhận dữ liệu từ máy tính hoặc Bluetooth có dạng như sau:



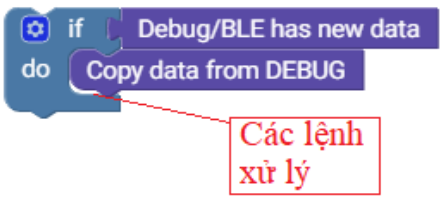
Hình 9.13: Cấu trúc lệnh nhận dữ liệu từ máy tính/điện thoại

Khi có dữ liệu mới được gửi đến từ máy tính hoặc điện thoại thì lệnh (1) sẽ có giá trị là "**true**" vì thế lệnh điều kiện nếu-thì được thực hiện. Khi đó các lệnh ở khu vực (2) sẽ được thực hiện. Khi không có dữ liệu mới vì lệnh điều kiện nếu-thì này không được thỏa mãn nên các lệnh xử lý dữ liệu bên trong không được thực hiện.

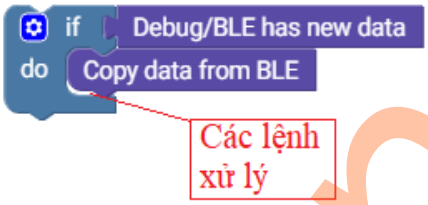
Khi đã phát hiện việc dữ liệu mới đến tiến trình cần thực hiện là sao chép dữ liệu nhận được vào biến **"Debug/Bluetooth Data"**. Lệnh này cần phải được thực hiện càng sớm càng tốt kể từ lúc phát hiện có dữ liệu mới. Vì sao vậy? Vì rất có thể đang có rất nhiều dữ liệu liên tục được gửi đi từ máy tính hoặc điện thoại đến Board. Nếu lệnh sao chép dữ liệu được thực hiện muộn có thể dữ liệu mới đến chiếm mất vị trí của dữ liệu cũ gây sai sót không mong muốn về xử lý và tính toàn vẹn của dữ liệu. Vì vậy, nguyên tắc là thực hiện sao chép dữ liệu nhận được vào biến sớm nhất có thể kể từ khi phát hiện được có dữ liệu mới.

Đối với hai luồng dữ liệu đến từ máy tính và đến từ điện thoại có cấu trúc khác nhau, vì vậy lệnh thực hiện sao chép dữ liệu của 2 luồng dữ liệu này cũng khác nhau. Tùy thuộc vào đối tượng giao tiếp mà cần sử dụng lệnh cho phù hợp.

Hai chương trình mẫu cho việc nhận và sao chép dữ liệu đến từ máy tính và điện thoại như sau:



Mẫu chương trình nhận dữ liệu từ máy tính



Mẫu chương trình nhận dữ liệu từ điện thoại

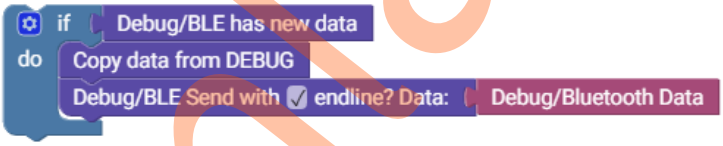
Hình 9.14: Mẫu chương trình nhận dữ liệu từ máy tính và điện thoại

Từ mẫu chương trình trên, có thể thêm một số lệnh xử lý dữ liệu cần thiết như kiểm tra nội dung bản tin, kiểm tra độ dài bản tin, ghép nội dung bản tin với bản tin khác hoặc đơn giản là gửi ngược lại nội dung của bản tin đến đối tượng đã gửi tin đến Board.

Ví dụ 3: Gửi chuỗi văn bản nhận được từ máy tính ngược trở về máy tính.

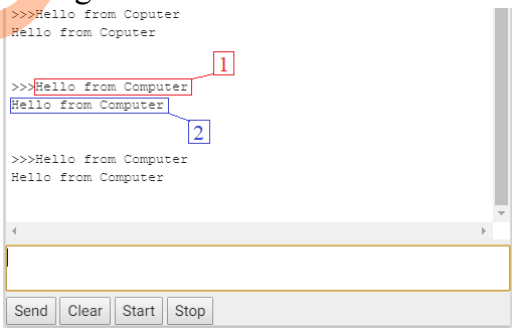
Phân tích: Với mẫu chương trình trên, khi có dữ liệu mới tổ hợp lệnh bên trong lệnh điều khiển nếu-thì sẽ được thực hiện và do đó lệnh sao chép dữ liệu vào biến **"Debug/Bluetooth Data"** được thực thi. Sau khi lệnh này thực thi, dữ liệu nhận được từ máy tính đã được sao chép sang biến này. Vì vậy, ở các lệnh xử lý dữ liệu chúng ta chỉ cần quan tâm đến biến này.

Trên cơ sở đó, chương trình nhận và gửi lại dữ liệu nhận được lên máy tính được xây dựng như sau:



Hình 9.15: Chương trình nhận và gửi lại dữ liệu về máy tính

Để thử nghiệm chương trình trên chúng ta mở giao diện **Debug Area** và bấm vào nút **Start** sau đó gõ một chuỗi văn bản bất kỳ rồi bấm vào nút **Send** trên phần mềm hoặc nhấn phím "**Enter**" trên bàn phím. Khi đó, dữ liệu được gửi đi từ máy tính qua USB đến Board. Khi Board nhận được dữ liệu từ máy tính sẽ thực hiện gửi ngược lại dữ liệu lên máy tính và dưới đây là kết quả của chương trình:

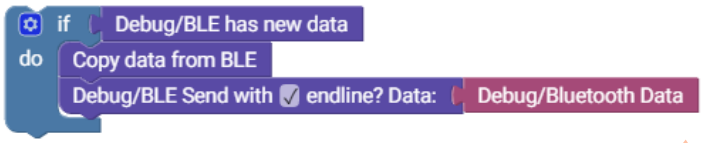


Hình 9.16: Kết quả thử nghiệm nhận và gửi lại dữ liệu về máy tính

1: Dữ liệu được gửi đi từ máy tính xuống Board.

2: Dữ liệu được gửi từ Board về máy tính

Ngay khi dữ liệu được gửi đi từ máy tính, Board sẽ gửi lại dữ liệu nhận được trong thời gian rất ngắn. Với kết quả này cho thấy chương trình giao tiếp cơ bản giữa máy tính và Board đã thành công. Chúng ta có thể tiến hành thử nghiệm với điện thoại sử dụng Terminal trên Clover Master với chương trình được thay đổi phù hợp như sau:



Hình 9.17: Chương trình nhận và gửi lại dữ liệu về điện thoại

3. Điều khiển thiết bị sử dụng lệnh điều khiển từ máy tính/điện thoại.

Từ kết quả giao tiếp truyền nhận chuỗi văn bản giữa máy tính/điện thoại với Board thành công, ta có thể sử dụng để thực hiện điều khiển Board từ máy tính hoặc điện thoại thông qua các dữ liệu gửi đến. Chúng ta cần quy định trước một số nội dung tương ứng thì tiến hành làm công việc tương ứng nào đó, sau đó chỉ cần truyền lệnh từ máy tính hoặc điện thoại xuống một cách tương ứng để điều khiển.

Với phần này ta tìm hiểu lệnh kiểm tra nội dung chuỗi nhận được có chứa một chuỗi ký tự nào đó không:

Hình 9.18: Cấu trúc lệnh kiểm tra nội dung chuỗi nhận được

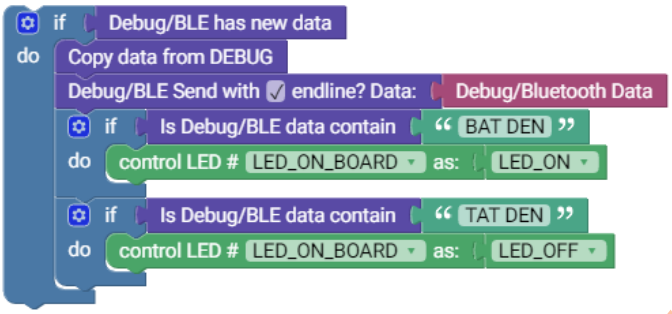
Lệnh trên dùng để kiểm tra xem nội dung chuỗi nhận được có chứa chuỗi "Hello" bên trong hay không? Nếu trong chuỗi nhận được có chuỗi "Hello" thì kết quả trả về sẽ là True (Đúng) ngược lại thì lệnh sẽ trả về kết quả là False (Sai).

Ví dụ: Nếu chuỗi văn bản nhận được là "Hello Clover" thì kết quả của lệnh trên là True. Nhưng nội dung nhận được là "H ello Clover" thì kết quả của câu lệnh trên là False.

Ví dụ 4: Xây dựng thiết bị điều khiển bật tắt đèn trên Board bằng lệnh điều khiển từ máy tính. Nếu nhận được lệnh "BAT DEN" thì tiến hành bật đèn trên Board, nếu nhận được lệnh "TAT DEN" thì tiến hành tắt đèn trên Board.

Với lệnh kiểm tra nội dung chuỗi ký tự ở trên ta có thể sử dụng để thực hiện chương trình này bằng cách nếu kiểm tra nội dung chuỗi nhận được có chứa chuỗi văn bản "BAT DEN" thì thực hiện lệnh điều khiển trạng thái đèn LED thành ON (Bật), nếu nội dung chuỗi nhận được có chứa chuỗi văn bản "TAT DEN" thì thực hiện lệnh điều khiển trạng thái đèn LED thành OFF (Tắt). Các trường hợp khác không phù hợp thì không thực hiện lệnh.

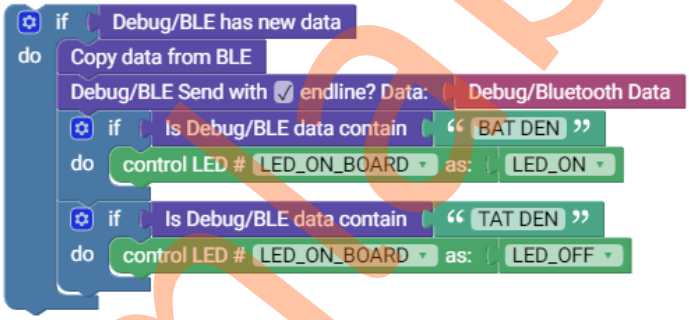
Chương trình được xây dựng như sau:



Hình 9.19: Chương trình điều khiển bật/tắt đèn bằng lệnh gửi từ máy tính

Tiến hành nạp chương trình vào Board và thử nghiệm chương trình này bằng cách lần lượt gõ lệnh "BAT DEN" và "TAT DEN" hoặc một số lệnh bất kỳ khác và gửi xuống Board để quan sát. Ta thấy Board đã hoạt động đúng với yêu cầu thiết kế ban đầu mà chúng ta đặt ra: Khi có lệnh "BAT DEN" thì đèn sáng, khi có lệnh "TAT DEN" thì đèn tắt, khi lệnh không chứa hai cụm từ trên thì trạng thái của đèn vẫn được giữ trạng thái cũ.

Từ chương trình này ta có thể tiến hành thay đổi nhỏ và tạo ra một thiết bị điều khiển bật tắt ánh sáng đèn bằng điện thoại. Chương trình như sau:



Hình 9.20: Chương trình điều khiển bật/tắt đèn bằng lệnh gửi từ điện thoại

Như vậy, với Clover Master và Clover Board chúng ta đã có thể thiết kế cho riêng mình một thiết bị điều khiển đèn hoặc thiết bị khác sử dụng điện thoại. Nếu muốn chỉ mình bạn có thể điều khiển được bạn có thể thay đổi các lệnh này theo một cấu trúc lệnh điều khiển mà chỉ riêng bạn biết và bạn sẽ làm chủ thiết bị đó.

Chú ý 1: Chương trình này có lệnh gửi dữ liệu ngược lại máy tính để kiểm thử đảm bảo chắc chắn Board đã nhận được dữ liệu hay chưa. Nếu Board gửi lại dữ liệu đã đúng với lệnh chúng ta đặt ra nhưng Board vẫn không đáp ứng lệnh khi đó chúng ta cần kiểm tra lại tư duy thiết kế chương trình. Đây cũng là một điểm chú ý khi chúng ta thiết kế các chương trình giao tiếp với máy tính hoặc điện thoại. Khi nào khẳng định chương trình đã hoạt động đúng theo yêu cầu thì có thể xóa bỏ lệnh gửi dữ liệu ngược lại đi (nếu không cần thiết) mà vẫn không ảnh hưởng đến chương trình.

Chú ý 2: Ngoài sử dụng cách gõ trực tiếp lệnh cần gửi vào Terminal trên điện thoại để gửi chúng ta có thể sử dụng các mục khác như Gamepad, Rotation hoặc Voice Control để gửi lệnh từ điện thoại xuống thiết bị. Ở đây để đơn giản chúng ta có thể sử dụng Voice

Control (Điều khiển bằng giọng nói) để điều khiển thiết bị mà không cần phải thực hiện công việc nhập lệnh điều khiển bằng bàn phím.

Như vậy ở phần này, chúng ta đã tìm hiểu được cách sử dụng chuỗi văn bản bằng CloverBlock, thực hiện giao tiếp 2 chiều (Gửi và nhận) thành công giữa Máy tính/Điện thoại với Board thiết bị từ đó có thể thực hiện những bài toán điều khiển của riêng mình để giải quyết các bài toán điều khiển thực tế.

Bài toán mở rộng: *Thiết kế chương trình đọc và gửi nhiệt độ, độ ẩm môi trường lên điện thoại và thực hiện bật tín hiệu cảnh báo theo điều khiển từ điện thoại. Cụ thể như sau:*

- *Tiến hành gửi về thông số nhiệt độ khi nhận được chuỗi văn bản "NHIET DO"*
- *Tiến hành gưi về thông số độ ẩm khi nhận được chuỗi văn bản "DO AM"*
- *Thực hiện bật còi cảnh báo khi nhận được chuỗi văn bản "NGUY HIEM"*
- *Thực hiện tắt còi cảnh báo khi nhận được chuỗi văn bản "AN TOAN"*

BÀI 10: LẬP TRÌNH ĐIỀU KHIỂN ROBOT CƠ BẢN

Ngoài được sử dụng để thiết kế chương trình điều khiển thiết bị hoặc đọc các thông số cảm biến, Clover còn hỗ trợ học lập trình thông qua xây dựng các chương trình cho Robot. Với mô hình Robot của Clover chúng ta có thể thiết kế Robot di chuyển theo quỹ đạo thiết kế sẵn, Robot điều khiển từ xa hoặc Robot tự hành đơn giản… và tùy theo khả năng sáng tạo của người sử dụng mà có thể thiết kế được Robot để giải quyết các bài toán khác nhau.

Các lệnh của CloverBlock về điều khiển Robot được liệt kê ở bảng sau:

STT	Block	Vị trí	Chức năng
1		Robot	Điều khiển tốc độ và chiều quay của mỗi động cơ độc lập. (Tốc độ quay từ 0-100%)
2		Robot	Cài đặt tốc độ di chuyển cho Robot (Tốc độ từ 0-100%)
3		Robot	Dừng di chuyển Robot
4		Robot	Điều khiển Robot tiến thẳng hoặc lùi thẳng
5		Robot	Điều khiển Robot rẽ trái hoặc rẽ phải (Có thể kèm vừa di chuyển tiến lùi vừa rẽ)
6		Robot	Điều chỉnh tốc độ di chuyển của Robot tăng hoặc giảm so với tốc độ hiện tại

Bảng 10.1: Các lệnh điều khiển Robot của CloverBlock

1. Điều khiển tốc độ và chiều động cơ

Điều khiển Robot di chuyển bản chất là phối hợp điều khiển quay 2 động cơ 2 bên của Robot quay với chiều và tốc độ phù hợp. Dưới đây là bảng chi tiết về phối hợp 2 động cơ để điều khiển Robot di chuyển.

Trong đó:

V_T là tốc độ quay của động cơ bên trái.

V_P là tốc độ quay của động cơ bên phải.

C_T là chiều của quay của động cơ trái.

C_P là chiều của quay của động cơ phải.

STT	Điều kiện	Mô tả di chuyển
1	$V_T = V_P$, C_T = Thuận, C_P = Thuận	Tiến thẳng
2	$V_T = V_P$, C_T = Ngược, C_P = Ngược	Lùi thẳng
3	$V_T > V_P$, C_T = Thuận, C_P = Thuận	Rẽ phải (Tiến và rẽ nếu V_P>0)
4	$V_T < V_P$, C_T = Thuận, C_P = Thuận	Rẽ trái (Tiến và rẽ nếu V_T>0)
5	$V_T > V_P$, C_T = Ngược, C_P = Ngược	Rẽ phải (Lùi và rẽ nếu V_P>0)
6	$V_T < V_P$, C_T = Ngược, C_P = Ngược	Rẽ trái (Lùi và rẽ nếu V_T>0)
7	$V_T = V_P$, $C_T \neq C_P$	Xoay tại chỗ
8	$V_T \neq V_P$, $C_T \neq C_P$	Di chuyển và xoay

Bảng 10.2: Mô tả các trạng thái điều khiển Robot

Lệnh điều khiển động cơ được mô tả chi tiết như sau:

Hình 10.1: Cấu trúc lệnh điều khiển động cơ

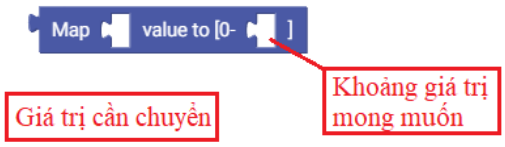
Trong đó động cơ điều khiển có thể là Motor1 (Động cơ số 1) hoặc Motor2 (Động cơ số 2). Chiều quay nếu được chọn là chiều quay thuận, không được chọn là chiều quay ngược. Tốc độ quay là số hoặc biến có giá trị trong khoảng từ 0-100.

Ví dụ 1: Điều khiển 1 động cơ quay với tốc độ thay đổi được bằng cách xoay biến trở và chiều động cơ có thể thay đổi bằng nút nhấn: Nút nhấn nhả thì động cơ quay theo chiều thuận, nút nhấn được nhấn thì động cơ quay theo chiều ngược.

Phân tích:

- Với chiều quay thay đổi được bằng việc thay đổi trạng thái nút nhấn ta có thể sử dụng một lệnh điều khiển điều kiện nếu-thì dạng 2 để đảm bảo cho cả 2 chiều quay của động cơ.
- Với việc thay đổi tốc độ động cơ: Giá trị đọc được của biến trở nằm trong khoảng từ 0-1023 trong khi giá trị tốc độ động cơ cho phép trong khoảng từ 0-100. Vì thế chúng ta cần chuyển đổi theo tỉ lệ tương đương. Với vấn đề này chúng ta có thể sử dụng cách lấy giá trị đọc được từ biến trở và chia cho 10.2 (1023:100=10.2) và sử dụng giá

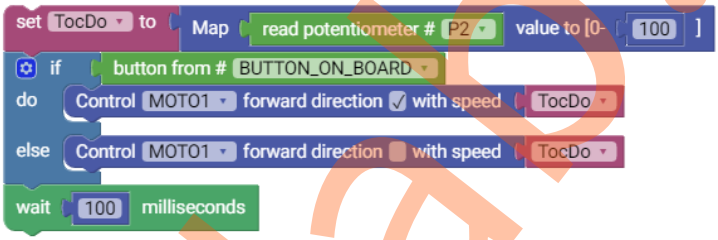
trị đó để điều khiển tốc độ động cơ. Hoặc có một cách khác mà CloverBlock đã hỗ trợ sẵn để chuyển đổi các giá trị từ 0-1023 về khoảng mong muốn bằng cách sử dụng lệnh "Map" của phần Math. Lệnh cụ thể như sau:



Hình 10.2: Cấu trúc lệnh chuyển đổi vùng giá trị

Trong đó giá trị cần chuyển nằm trong khoảng từ 0-1024 và khoảng giá trị mong muốn là giá trị lớn nhất muốn chuyển đổi. Với bài toán này tương ứng với tốc độ cho phép lớn nhất của động cơ là 100.

Từ những phân tích trên ta xây dựng được chương trình điều khiển giải quyết bài toán trên như sau:



Hình 10.3: Chương trình thử nghiệm điều khiển động cơ

Bước 1: Giá trị của biến trở được đọc rồi chuyển đổi sang khoảng từ 0-100 để phù hợp với tốc độ điều khiển độc cơ và lưu vào biến có tên là "TocDo".

Bước 2: Kiểm tra trạng thái của nút.

- Nếu nút nhấn ở trạng thái nhả (Không được nhấn) thì thực hiện điều khiển động cơ quay thuận với tốc độ đã được tính toán ở trên.
- Nếu nút nhấn ở trạng thái nhấn thì thực hiện điều khiển động cơ quay theo chiều ngược với tốc độ đã được tính toán ở trên.

Bước 3: Tạm dừng chương trình 100ms trước lần cập nhật tốc độ và chiều tiếp theo.

Tiến hành nạp chương trình vào Board, kết nối các thiết bị theo đúng hướng dẫn hiển thị trên CloverBlock và tiến hành thử nghiệm. Trước tiên, chúng ta thử xoay biến trở để thay đổi tốc độ thay giá trị xoay biến trở. Sau khi đã thử nghiệm thay đổi tốc độ thành công chúng ta tiến hành thay đổi trạng thái của nút nhấn để xem xét chiều quay động cơ có thay đổi hay không.

Như vậy, với ví dụ trên chúng ta đã nắm được cách điều khiển từng động cơ độc lập. Với nền tảng này chúng ta có thể kết hợp 2 động cơ để tạo ra sự di chuyển cho Robot theo mong muốn. Phần tiếp theo sẽ giới thiệu về điều khiển Robot di chuyển.

2. Điều khiển Robot di chuyển một số hướng cơ bản

Với hai lệnh điều khiển động cơ động lập có thể kết hợp tạo ra được các cách di chuyển khác nhau. Tuy nhiên, để đơn giản hóa quá trình điều khiển Robot CloverBlock cung cấp một số lệnh điều khiển Robot cơ bản về di chuyển và hướng để người sử dụng có thể nhanh chóng thiết kế được một Robot di chuyển theo ý muốn.

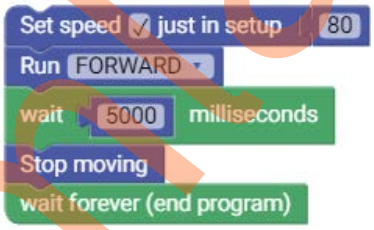
Phần này sử dụng các hàm điều khiển di chuyển và hướng của Robot để tạo ra Robot di chuyển theo một số cách thức quy định trước.

Ví dụ 2: Điều khiển Robot tiến thẳng với vận tốc 80% trong 5 giây sau đó dừng lại.

Phân tích: Với các lệnh đã được nhắc đến ở bảng 9.1, ta có các lệnh về điều khiển di chuyển thẳng, di chuyển theo hướng và điều khiển dừng Robot. Ở ví dụ này chúng ta sử dụng các lệnh đó để việc xây dựng chương trình nhanh và gọn gàng hơn.

Để Robot có thể di chuyển bước đầu tiên cần tiến hành cài đặt tốc độ động cơ ở giá trị 80%. Vì khi di chuyển Robot không thay đổi tốc độ nên chúng ta chọn tốc độ này chỉ cài đặt một lần khi khởi động. Sau đó, thực hiện lệnh chạy tiến thẳng. Để đảm bảo Robot được chạy thẳng trong 5 giây cần thực hiện tạm dừng chương trình trong 5000ms để trong quá trình đó 2 động cơ vẫn quay đều ở tốc độ 80%. Sau khi đủ 5 giây, sử dụng lệnh dừng lại để dừng sự di chuyển của Robo. Và cuối cùng thêm lệnh "**wait forever**" để đảm bảo chương trình không quay lại chạy lệnh đầu tiên của chương trình sẽ làm Robot tiếp tục di chuyển.

Từ đó có chương trình điều khiển như sau:

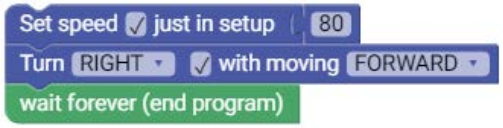


Hình 10.4: Chương trình điều khiển Robot di chuyển thẳng

Tiến hành thử nghiệm chương trình trên và đánh giá kết quả. Thực tế cho thấy Robot đã di chuyển thẳng trong 5s và dừng lại theo yêu cầu của bài toán.

Ví dụ 3: Điều khiển Robot di chuyển tạo thành hình tròn.

Phân tích: Để Robot di chuyển tạo thành hình tròn cần tiến hành cho Robot vừa tiến (hoặc lùi) vừa rẽ theo một hướng cố định. Việc này giống như khi chúng ta lái xe và bẻ lái thì phương tiện sẽ tạo thành hình vòng cung. Nếu tiếp tục giữ tay lái ở góc đó đủ lâu chúng ta thấy phương tiện quay về vị trí xuất phát. Áp dụng ý tưởng đó để xây dựng chương trình điều khiển Robot di chuyển theo hình tròn. Chương trình điều khiển Robot như sau:

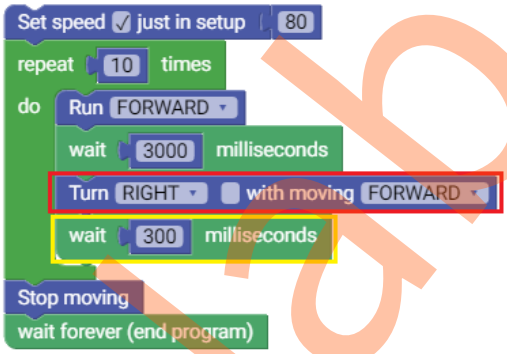


Hình 10.5: Chương trình điều khiển Robot di chuyển tròn

Công việc duy nhất cần thực hiện là cho Robot di chuyển và rẽ theo 1 hướng cố định. Vì vậy, chương trình đơn giản chỉ cần cài đặt tốc độ sau đó di chuyển liên tục. Vì vậy, Robot sẽ liên tục di chuyển tạo thành hình tròn trong suốt quá trình di chuyển.

Ví dụ 4: Điều khiển Robot di chuyển tạo thành hình vuông. Mỗi cạnh di chuyển trong 3 giây.

Phân tích: Để Robot di chuyển thành hình vuông cần thực hiện cho Robot di chuyển tiến thẳng trong 1 khoảng thời gian sau đó dừng lại để quay trái (Hoặc phải) sang cạnh di chuyển thứ hai. Sau đó thực hiện lặp lại thao tác di chuyển đó thêm 3 lần nữa để đảm bảo Robot có thể quay về được vị trí ban đầu. Như vậy, ở đây ta cần sử dụng kết hợp 2 hàm di chuyển tiến thẳng và rẽ trái (hoặc rẽ phải) lần lượt với nhau trong 4 lần. Vì 4 lần này các động tác di chuyển là giống nhau nên ta sử dụng vòng lặp để chương trình nhìn ngắn gọn hơn. Từ những phân tích đó ta xây dựng được chương trình như sau:



Hình 10.6: Chương trình điều khiển Robot di chuyển tạo thành hình vuông

Ta thấy lệnh rẽ phải (Viền đỏ) không được tích chọn cùng với di chuyển để đảm bảo Robot thực hiện quay tạo chỗ nhằm tạo ra góc vuông làm cơ sở để tạo ra hình vuông. Nếu ta tích chọn cùng với di chuyển thì tại vị trí này sẽ vừa rẽ vừa di chuyển và sẽ tạo một hình vòng cung thay vì góc vuông. Ngoài ra, ở lệnh tạm dừng chương trình sau lệnh rẽ phải được tô vàng. Đây là giá trị thời gian để đảm bảo Robot quay được góc vuông. Với mỗi Robot thời gian này có thể có sự sai khác, vì vậy chúng ta cần thử nghiệm và tìm ra giá trị thời gian phù hợp để Robot của chúng ta có thể di chuyển tạo ra một hình vuông tốt nhất (Vị trí dừng lại gần sát với vị trí xuất phát nhất).

Tiến hành nạp chương trình vào Robot rồi thử nghiệm, sau đó trong quá trình thử nghiệm có thể thay đổi giá trị thời gian rẽ phải để đảm bảo Robot tạo được hình vuông phù hợp nhất. Chúng ta nên ghi nhớ giá trị này để sử dụng cho các lần sau nếu cần thiết.

3. Điều khiển khởi động và dừng Robot bằng điều khiển hồng ngoại

Ở các ví dụ trên ta thấy sau khi nạp chương trình nếu động cơ đang kết nối vào Board điều khiển thì Robot sẽ ngay lập tức di chuyển đôi khi gây ra phiền phức và thiếu chủ động. Ở phần này chúng ta sẽ sử dụng điều khiển hồng ngoại để điều khiển Robot di chuyển và dừng theo ý muốn.

Với bài toán này chúng ta tìm hiểu cách sử dụng điều khiển hồng ngoại để điều khiển Robot nói riêng và có thể áp dụng điều khiển thiết bị nói chung.

Dưới đây là bảng các lệnh sử dụng với điều khiển hồng ngoại trên CloverBlock:

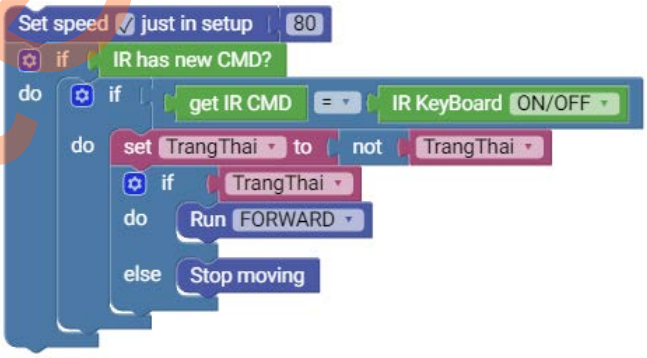
STT	Block	Vị trí	Chức năng
1		Input	Kiểm tra có dữ liệu vừa được gửi đến từ điều khiển hồng ngoại đến Board hay không
2		Input	Lấy giá trị lệnh mới được gửi đến. Giá trị lệnh đọc về là số nguyên.
3		Input	Mã của các lệnh tương ứng trên điều khiển hồng ngoại đi kèm.

Bảng 10.3: Các lệnh giao tiếp với điều khiển hồng ngoại

Ví dụ 5: Xây dựng chương trình điều khiển Robot khi nhận được lệnh điều khiển từ nút ON/OFF trên điều khiển hồng ngoại thì thực hiện di chuyển tạo thành hình tròn, nếu nhấn nút ON/OFF một lần nữa thì Robot thực hiện dừng lại.

Phân tích: Ta thấy chỉ với một nút nhấn nhưng chúng ta phải điều khiển Robot ở 2 trạng thái khác nhau: Nếu đang dừng thì di chuyển và nếu đang di chuyển thì dừng lại. Điều này hoàn toàn tương tự với các thiết bị trong gia đình chúng ta sử dụng như điều khiển bật tắt tivi, điều hòa… Vậy làm các nào để giải quyết tình huống này.

Giải pháp ở đây là sử dụng một biến để lưu giữ trạng thái đang di chuyển. Vì Robot trong bài toán này chỉ có hai trạng thái là di chuyển và dừng nên ta có thể dùng một biến dạng Boolean (Chỉ True và False) làm biến lưu giữ trạn thái di chuyển. Giả sử khi biến này mang giá trị False thì Robot đứng yên và nếu biến này mang giá trị True thì Robot di chuyển. Và mỗi lần nhận được tín hiệu nút ON/OFF được nhấn từ điều khiển hồng ngoại thì thực hiện thay đổi giá trị này từ True thành False hoặc từ False thành True. Việc thay đổi giá trị này có thể được thực hiện được bằng lệnh điều khiển điều kiện nếu-thì hoặc đơn giản hơn là sử dụng lệnh "**not**" (lệnh đảo). Sau mỗi lần giá trị biến trạng thái này thay đổi ta cho phép lệnh điều khiển Robot di chuyển được cập nhật theo. Như vậy chúng ta đã xây dựng được một chương trình điều khiển Robot di chuyển theo lệnh điều khiển hồng ngoại. Chương trình cụ thể như sau:



Hình 10.7: Chương trình điều khiển Robot di chuyển dùng điều khiển hồng ngoại

Giống với giao tiếp với máy tính hoặc điện thoại chương trình cần liên tục kiểm tra liệu có dữ liệu mới đến từ điều khiển hồng ngoại hay không. Bàn phím điều khiển hồng ngoại

có rất nhiều phím vì thế cần thực hiện kiểm tra mã nhận được có đúng là mã của phím ON/OFF hay không. Nếu không phải thực hiện bỏ qua và chờ lệnh điều khiển mới. Còn nếu lệnh đúng là đến từ phím ON/OFF thì thực hiện các lệnh liên quan thay đổi giá trị của biến và trạng thái di chuyển của Robot.

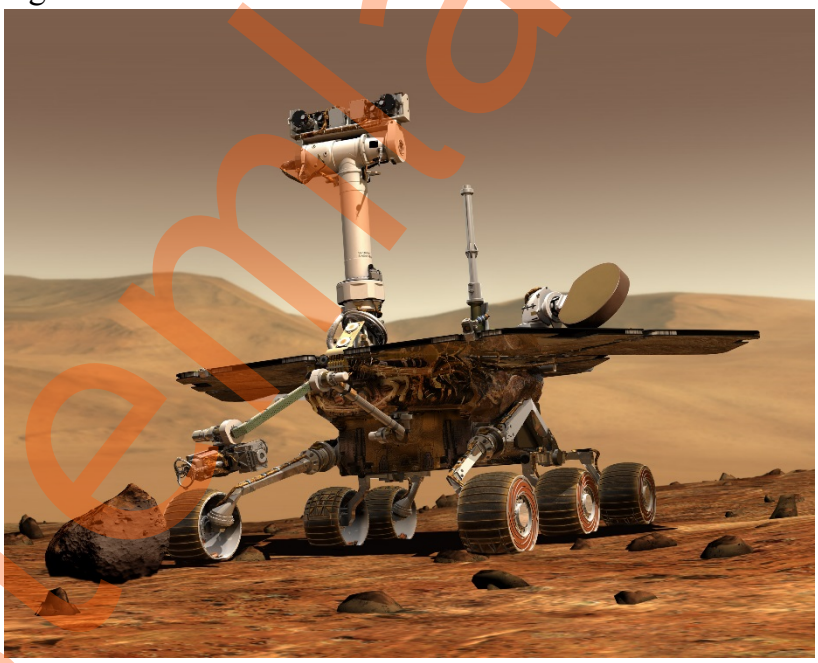
Như đã phân tích ở trên, ta sử dụng biến "*TrangThai*" đại diện cho trạng thái di chuyển của Robot. "*TrangThai*" bằng True thì di chuyển và "*TrangThai*" bằng False thì dừng lại. Vì vậy, ta đặt hai lệnh di chuyển và dừng lại vào 2 khu vực lệnh thực thi của lệnh điều khiển điều kiện nếu-thì dạng 1 để cập nhật trạng thái di chuyển vào Robot.

Chương trình này là một chương trình được chạy liên tục nên chúng ta không cần sử dụng lệnh "**wait forever**" để đảm bảo trong suốt quá trình được cấp nguồn Robot sẽ lắng nghe lệnh từ điều khiển hồng ngoại và di chuyển phù hợp với lệnh điều khiển.

Thực hiện nạp và thử nghiệm chương trình trên Robot. Từ đây, chúng ta có thể thực hiện tạo ra nhiều chương trình điều khiển khác mà chúng ta thấy là cần thiết.

4. Xây dựng Robot tự động tránh vật cản

Những hành động, di chuyển ở những phần trước của Robot đều là những di chuyển một cách thụ động hoặc là theo một số cách di chuyển có sẵn hoặc là nhận lệnh để di chuyển theo yêu cầu. Ở phần này, chúng ta sẽ thiết kế một chương trình cho Robot để biến Robot trở nên thông minh hơn.



Hình 10.8: Hình ảnh Robot khám phá trên sao Hỏa

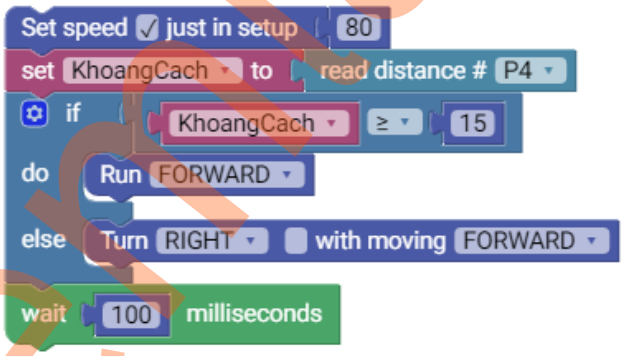
Trong thực tế các Robot được phóng lên các hành tinh khác như sao Hỏa, mặt trăng để thăm dò hoặc các tàu vũ trụ thăm dò không gian không thường xuyên nhận được lệnh điều khiển từ con người do thời gian để gửi một lệnh từ trái đất đến tàu vũ trụ hoặc Robot đó mất thời gian tương đối dài. Vì vậy, người điều khiển ở trạm điều khiển chỉ đưa ra các lệnh điều khiển chính theo nhiệm vụ, còn việc thực hiện trực tiếp sẽ do Robot "quan sát", "thăm dò" để hiểu được môi trường xung quanh và đưa ra lệnh điều khiển phù hợp.

Với những thiết bị đang có trong tay và với những kiến thức đang có chúng ta không đủ khả năng để thiết kế được một Robot hoạt động như thế nhưng chúng ta sẽ thiết kế một Robot đơn giản theo cách tương đương để khám phá về các Robot tự hành.

Ví dụ 6: Thiết kế Robot di chuyển đi thẳng và tự động tránh vật cản để tìm đường đi khác nhằm phục vụ khám phá các địa hình khu vực chưa được biết đến.

Phân tích: Ở bài toán này chúng ta sử dụng cảm biến khoảng cách để đo khoảng cách Robot đến hướng di chuyển phía trước. Nếu khoảng cách đo được nhỏ hơn một ngưỡng nào đó (Ngưỡng khoảng cách an toàn) thì thực hiện điều khiển Robot tránh vật cản bằng cách di chuyển rẽ sang một hướng khác (Tùy thuộc vào lựa chọn của người lập trình) hoặc di chuyển rẽ sang một hướng ngẫu nhiên rồi tiếp tục di chuyển thẳng để khám phá hướng mới. Nếu khoảng cách an toàn thì vẫn tiếp tục di chuyển thẳng và chỉ rẽ khi khoảng cách an toàn đến vật cản bị vi phạm.

Từ ý tưởng trên, tiến hành xây dựng chương trình điều khiển cho Robot tự động tránh vật cản. Trước tiên, cần cài đặt tốc độ di chuyển cho Robot để Robot có thể di chuyển như đã nhắc ở trên. Sau đó, liên tục đọc khoảng cách từ Robor đến vật cản phía trước. Sử dụng giá trị khoảng cách đọc được này so sánh với giá trị khoảng cách an toàn mong muốn. Nếu khoảng cách an toàn được đảm bảo thì lệnh di chuyển thẳng được thực hiện. Nếu khoảng cách an toàn bị vi phạm thì thực hiện tiến hành cho Robot rẽ sang bên phải (Tùy chọn). Quy trình được lặp lại liên tục để Robot tiến hành khám phá môi trường mới một cách ngẫu nhiên. Chương trình cụ thể như sau:



Hình 10.9: Chương trình Robot tự động tránh vật cản

Ví dụ này, quy định khoảng cách an toàn là 15cm (Có thể lựa chọn giá trị khác). Chúng ta sử dụng lệnh điều khiển điều kiện nếu-thì dạng 1 để có thể điều khiển Robot tiến thẳng hoặc rẽ phải trong hai điều kiện khoảng cách an toàn và không an toàn. Giữa các lần cập nhật hướng di chuyển chương trình đợi tạm dừng 100ms để Robot có thời gian di chuyển hoặc rẽ để đảm bảo khoảng cách ở lần kế tiếp được thay đổi. Chương trình lặp lại liên tục như vậy và Robot sẽ tự động khám phá môi trường xung quanh mà không cần sự phải can thiệp điều khiển hoặc cài đặt hướng di chuyển cố định nào trước.

Như vậy, với ví dụ trên chúng ta đã hoàn thiện được một Robot có khả năng tự động thăm dò và khám phá môi trường xung quanh ở mức độ đơn giản. Chúng ta có thể sử dụng

nhiều hơn các cảm biến phối hợp để xây dựng được các Robot khám phá môi trường có tính năng cao cấp hơn.

Ví dụ mở rộng 1: *Sử dụng cảm biến dò vạch để tiến hành điều khiển Robot di chuyển tự động theo đường vạch đã kẻ trên đường.*

Ví dụ mở rộng 2: *Mở rộng từ ví dụ số 1, ở hai đầu của đường đã được kẻ sẵn có 2 ô vị trí có kích thước lớn. Thực hiện lập trình Robot khi đến 2 đầu di chuyển này thì dừng lại để các Robot khác thực hiện bốc/dỡ hàng sau đó tự động quay đầu về đầu còn lại để các Robot ở đầu bên kia thực hiện dỡ/bốc hàng lên Robot.*

Hình 10.10: Hình ảnh Robot chở hàng tự động trong nhà máy

Ví dụ mở rộng 3: *Từ hình ảnh trên, mở rộng ví dụ mở rộng số 2. Thiết kế Robot tự động di chuyển phục vụ bốc dỡ hàng có khả năng tự động dừng lại nếu phái trước có vật cản nhằm nâng cao an toàn lao động trong nhà máy. (Khi xuất hiện vật cản bất thường phía trước Robot tự động dừng lại cho đến khi vật cản di chuyển qua mới tiến hành di chuyển tiếp).*

BÀI 11: LẬP TRÌNH ROBOT CÓ ĐIỀU KHIỂN

Bài số 8 và bài số 9 đã giới thiệu cách giao tiếp giữa Board với điện thoại thông qua giao tiếp Bluetooth và các lệnh điều khiển Robot di chuyển cơ bản. Ở phần này, chúng ta sẽ sử dụng kết hợp kết hợp kết quả của hai bài học trên tiến hành xây dựng một số chương trình về điều khiển Robot sử dụng phần mềm Clover Master trên điện thoại như: Điều khiển bằng thao tác điều khiển phím, điều khiển bằng cảm biến góc nghiêng của điện thoại và điều khiển Robot di chuyển bằng giọng nói.

1. Robot điều khiển bằng giọng nói (Voice Control)

Ví dụ 1: Xây dựng chương trình điều kiển Robot di chuyển bằng giọng nói theo yêu cầu như sau:

- Nếu người điều khiển nói “Tiến lên” thì di chuyển tiến thẳng.
- Nếu người điều khiển nói “Lùi lại” thì di chuyển lùi thẳng.
- Nếu người điều khiển nói “Sang trái” thì rẽ trái 1 góc 90 độ rồi di chuyển thẳng.
- Nếu người điều khiển nói “Sang phải” thì rẽ phải 1 góc 90 độ rồi di chuyển thẳng.
- Nếu người điều khiển nói “Dừng lại” thì dừng di chuyển.

Phân tích:

Về phần mềm Clover Master: Clover Board và Clover Master thực hiện giao tiếp dữ liệu dưới dạng các ký tự không có dấu, vì vậy khi nhận được lệnh điều khiển bằng giọng nói của người nói phần mềm tự động thực hiện chuyển đổi sang tiếng việt không dấu và biến tất cả các chữ cái viết thường thành viết hoa sau đó mới gửi đến Board thông qua Bluetooth.

Ví dụ: Nếu người dùng nói: “Tiến lên” thì Board sẽ nhận được dữ liệu là “TIEN LEN”

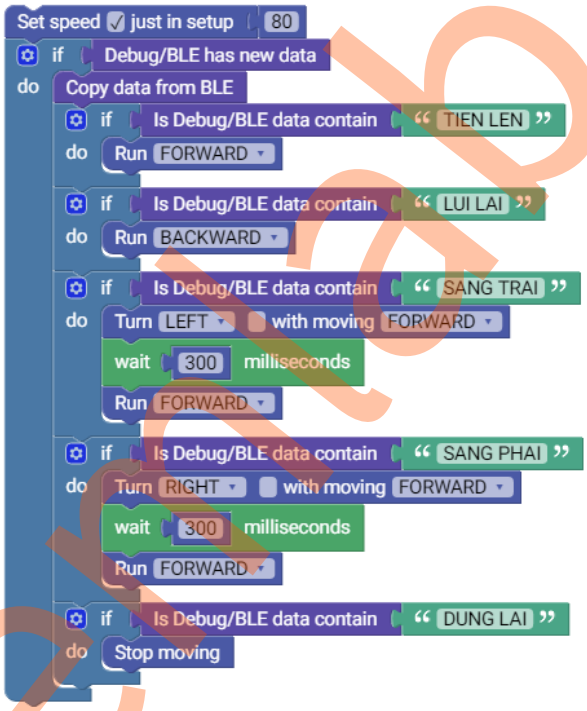
Tương tự như vậy chúng ta có lệnh thực tế mà Board nhận được khi có lệnh điều khiển giọng nói theo yêu cầu của ví dụ như sau:

Lệnh nói	Lệnh nhận được	Hành động
Tiến lên	TIEN LEN	Di chuyển tiến thẳng
Lùi lại	LUI LAI	Di chuyển lùi thẳng
Sang trái	SANG TRAI	Rẽ trái rồi di chuyển thẳng
Sang phải	SANG PHAI	Rẽ phải rồi di chuyển thẳng
Dừng lại	DUNG LAI	Dừng di chuyển

Bảng 11.1: Bảng lệnh điều khiển giọng nói

Từ ví dụ điều khiển thiết bị bằng lệnh điều khiển qua máy tính/điện thoại (bài số 8) ta tiếp tục phát triển để phù hợp với yêu cầu của ví dụ này. Thay vì sử dụng các lệnh điều khiển bật tắt đèn, ta sẽ thay thế bằng các lệnh điều khiển di chuyển Robot phù hợp (đã được giới thiệu chi tiết ở bài số 9) để Robot di chuyển đúng theo yêu cầu cả đề bài.

Chương trình cho Robot được xây dựng như sau:



Hình 11.1: Chương trình điều khiển Robot sử dụng giọng nói

Để tiến hành thử nghiệm chương trình này, tiến hành nạp chương trình vào Robot sau đó sử dụng phần mềm Clover Master ở phần Voice Control để điều khiển. Để điều khiển chúng ta thực hiện nói các lệnh tương ứng ở bảng trên để điện thoại gửi đến Robot. Khi nhận được các lệnh tương ứng Robot sẽ đáp ứng các lệnh đó và di chuyển theo cách mà chúng ta đã lập trình ở trên. Giao diện điều khiển bằng giọng nói có giao diện như sau:

Hình 11.2: Giao diện điều khiển bằng giọng nói của phần mềm Clover Master

Từ chương trình trên ta có thể thay đổi các lệnh tương ứng thành các lệnh tương ứng khác hoặc phát triển chương trình với nhiều lệnh và cơ chế điều khiển khác như xoay tại chỗ, chạy hình vòng tròn, tăng tốc, giảm tốc,….để tạo ra những Robot theo lệnh điều khiển mang tính cá nhân và sáng tạo hơn.

2. Robot điều khiển bằng cảm biến góc nghiêng (Rotation)

Ví dụ 2: Xây dựng chương trình điều khiển Robot bằng cảm biến góc nghiêng của điện thoại.

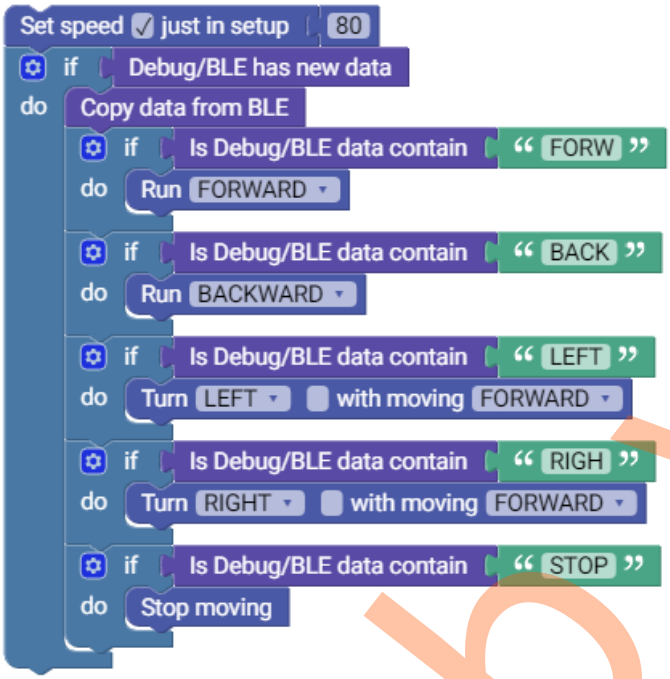
Phân tích: Bảng các lệnh được gửi từ phần Rotation của Clover Master được quy định như sau:

Lệnh nhận được	Hành động
ROTFORW	Di chuyển tiến thẳng
ROTBACK	Di chuyển lùi thẳng
ROTLEFT	Rẽ trái
ROTRIGH	Rẽ phải
ROTSTOP	Dừng di chuyển

Bảng 11.2:Bảng lệnh điều khiển sử dụng Rotation của Clover Master

Khi ở chế độ Rotation, chúng ta chỉ cần nghiêng điện thoại theo các hướng di chuyển mong muốn thì điện thoại sẽ gửi đi các bản tin tương ứng như bảng trên trong đó cụm từ "ROT" đại diện cho phần "ROTATION". Chương trình ở bài toán này tương đương với bài toán ở ví dụ điều khiển bằng giọng nói và có khác biệt một nhỏ về chuỗi nhận được để điều khiển và các lệnh rẽ.

Từ những thông tin trên ta xây dựng chương trình điều khiển như sau:



Hình 11.3: Chương trình điều khiển Robot sử dụng giao diện Rotation của Clover Master

Với chương trình trên chúng ta sử dụng phần Rotation của Clover Master để tiến hành thử nghiệm. Tiến hành thay đổi góc nghiêng điện thoại ở các góc khác nhau và kiểm thử bằng cách quan sát theo dõi quá trình di chuyển tương ứng của Robot. Giao diện điều khiển Rotation của Robot như sau:

Hình 11.4: Giao diện Rotation của phần mềm Clover Master

3. Robot điều khiển bằng phím điều khiển (Gamepad)

Ví dụ 3: Xây dựng chương trình điều khiển Robot di chuyển theo điều khiển GamePad của phần mềm Clover Master, với bảng chuỗi ký tự tương ứng nhận được cụ thể như sau:

Ký hiệu	Lệnh nhận được	Hành động
	PADSTOP	Dừng di chuyển
	PADFORW	Di chuyển tiến thẳng
	PADBACK	Di chuyển lùi thẳng
	PADLEFT	Rẽ trái
	PADRIGH	Rẽ phải
	PADFORW_LEFT	Di chuyển tiến kèm rẽ trái
	PADFORW_RIGH	Di chuyển tiến kèm rẽ phải
	PADBACK_LEFT	Di chuyển lùi kèm rẽ trái
	PADBACK_RIGH	Di chuyển lùi kèm rẽ phải
	PADSSSS	(Tùy biến)
	PADTTTT	(Tùy biến)
	PADXXX	(Tùy biến)
	PADOOOO	(Tùy biến)

Bảng 11.3: Bảng lệnh điều khiển của giao diện Gamepad (Clover Master)

Từ bảng trên chúng ta dự kiến thiết kế chương trình có khả năng di chuyển theo 8 hướng: Tiến, lùi, rẽ trái, rẽ phải, tiến và rẽ trái, tiến và rẽ phải, lùi và rẽ trái, lùi và rẽ phải và một lệnh để dừng lại. Ngoài ra, với 4 nút có chức năng tùy biến chúng ta có thể tận dụng để điều chỉnh tốc độ di chuyển của Robot. Chúng ta chọn nút ô vuông làm nút tăng tốc và nút tam giác để giảm tốc độ.

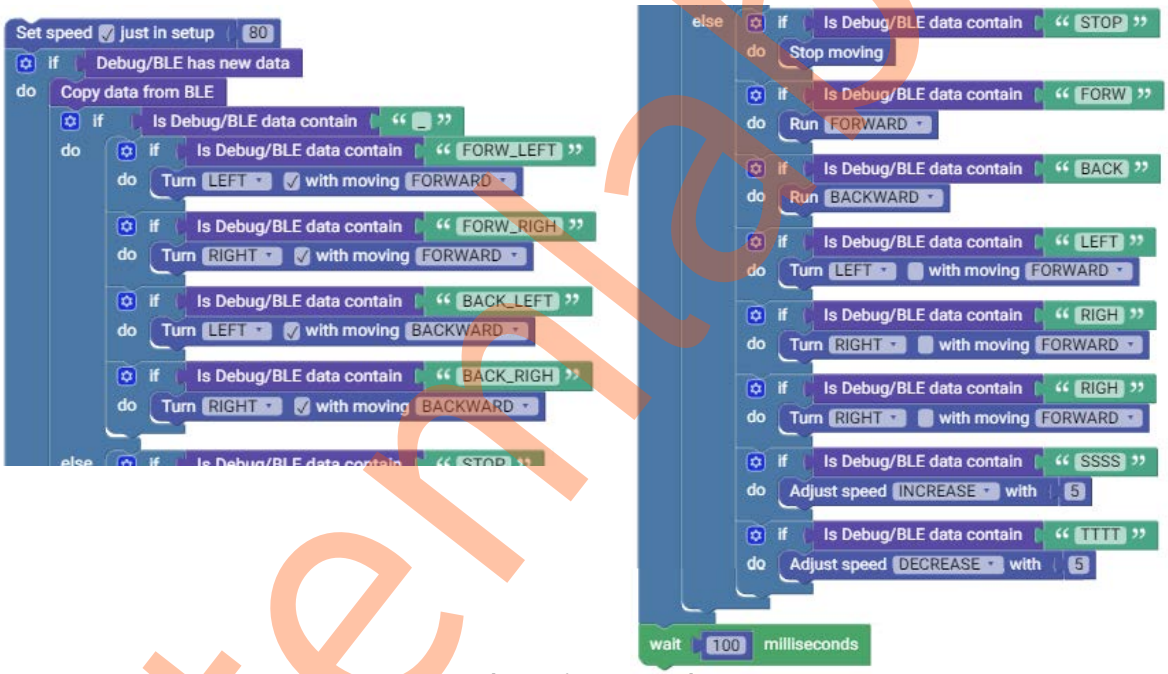
Ta thấy ở các hướng di chuyển kết hợp có nội dung gửi đến Board là tổng hợp 2 chuỗi văn bản của chiều di chuyển và chiều rẽ. Nếu chương trình vẫn được xử lý như 2 ví dụ trên thì sẽ

xảy ra trường hợp khi lệnh điều khiển di chuyển kết hợp được gửi đi thì có 3 lệnh được thực thi. Cụ thể như ví dụ sau:

Lệnh điều khiển gửi đi là "PADFORW_LEFT" thì trong bản tin bao gồm cả chuỗi văn bản "FORW" và "LEFT". Nếu chương trình được xây dựng sử dụng cách kiểm tra nội dung văn bản cũ thì xảy ra hiện tượng ngoài lệnh vừa di chuyển vừa rẽ trái được thực thi thì các lệnh tiến thẳng (FORW) và rẽ trái không di chuyển (LEFT) cũng được thực hiện. Và đây là một vấn đề không mong muốn. Vậy giải pháp là gì?

Từ bảng nội dung gửi đi ta thấy các bản tin điều khiển di chuyển kết hợp xuất hiện một ký tự mà các lệnh khác không có đó là ký tự "_" (Gạch dưới). Ta có thể tận dụng ký tự gạch dưới "_" để phân biệt được lệnh điều khiển kết hợp và lệnh điều khiển đơn hướng nhằm tránh tình trạng cách lệnh được thực hiện ngoài ý muốn.

Từ những phân tích đó,chương trình điều khiển Robot được thiết kế cụ thể như sau. Vì nội dung chương trình tương đối dài nên được cắt làm 2 phần để hiển thị:

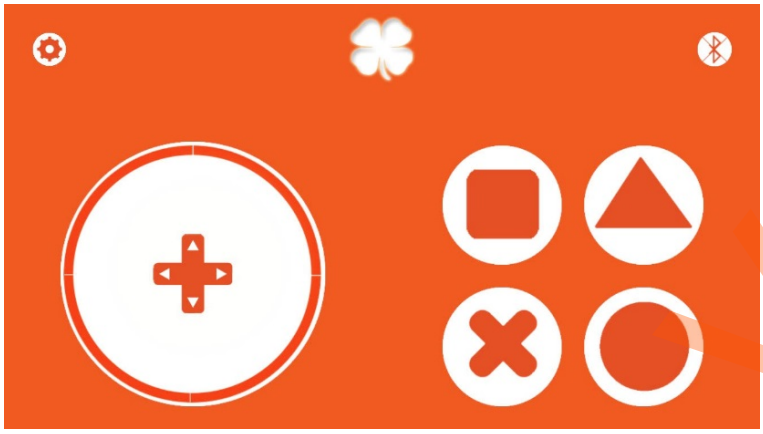


Hình 11.5: Chương trình điều khiển Robot bằng Gamepad của Clover Master

Do có nhiều lệnh được gửi ở các trường hợp khác nhau nên chương trình tương đối dài và phức tạp. Tuy nhiên, về cách giải quyết chung thì không khác nhiều với những bài toán trước đây. Ở bài toán này chúng ta chỉ sử dụng nhiều lệnh kiểm tra hơn các trường hợp còn lại và sử dụng kết hợp với các lệnh điều khiển Robot phù hợp. Tương tự như vậy, ở mức độ các bài toán khác chúng ta có thể sử dụng nhiều lệnh để điều khiển nhiều thiết bị với nhiều chế độ khác nhau.

Với ví dụ này, chúng ta sử dụng phần Gamepad của Clover Master để tiến hành điều khiển và thử nghiệm Robot. Kết quả cho thấy với một chương trình trở nên phức tạp chúng ta đã có

thể điều khiển Robot di chuyển với nhiều tính năng khác nhau về các hướng và cả tốc độ. Giao diện Gamepad của Clover Master như sau:



Hình 11.6: Giao diện điều khiển Gamepad của Clover Master

Ví dụ mở rộng: *Xây dựng Robot điều khiển bằng Gamepad như ví dụ trên tự động dừng khi khoảng cách tới vật cản phía trước ở dưới ngưỡng an toàn (Bật còi cảnh báo và tự động dừng lại).*

LỜI KẾT

Mục đích chính của cuốn sách là góp phần giúp những người không chuyên đặc biệt là các bạn học sinh tiếp cận lập trình và STEM bằng phương thức đơn giản, hạn chế sự phức tạp của cấu trúc chương trình của các ngôn ngữ lập trình thông thường. Cuốn sách đã lần lượt giới thiệu được những kiến thức, vấn đề cơ bản nhất để bắt đầu làm quen với lập trình thông qua phần mềm Clover Block, Clover Board và Clover Master.

Với sự đóng góp nhỏ bé của mình, đội ngũ của Clover Team hi vọng sẽ góp phần khích lệ được sự đam mê học hỏi, nghiên cứu công nghệ của các bạn trẻ - thế hệ tương lai của đất nước, từ đó góp phần vào sự phát triển chung của nền công nghệ nước nhà.

Clover Team rất vui lòng đón nhận những lời góp ý của quý bạn đọc thông qua địa chỉ email: robolabcenter@gmail.com

Xin chân thành cảm ơn!

VỀ NHÓM TÁC GIẢ

Cuốn sách được hoàn thành bởi nhóm tác giả của Clover Team, bao gồm:

1. Lương Công Duẩn
2. Trương Minh Đức – CEO & Co-Founder của Clover
3. Lê Xuân Đạo – Kỹ sư thiết kế làm việc lại Clover
4. Các thành viên của Clover Team.

Ngoài ra, cuốn sách còn nhận được rất nhiều sự đóng góp và cố vấn của các thầy cô giáo đang công tác tại Khoa Kỹ thuật Điện tử I - Học viện Công nghệ Bưu chính Viễn thông.

GIẤY PHÉP SỬ DỤNG TÀI LIỆU