Trường Đại Học Bách Khoa Hà Nội

Khoa Công nghệ thông tin và Truyền thông

Báo cáo Thực hành Kiến trúc máy tính

Giảng viên hướng dẫn
ThS. Lê Bá Vui

Nhóm: 6
Ngô Thị Lam - 20205210 - Bài 5
Trần Vinh Khánh - 20205208 - Bài 1

Ngày 21 tháng 7 năm 2022

Mục lục

Danh sách các Thuật Toán

Danh sách hình vẽ

Bài 1
Curiosity Marsbot

1.1 Đề bài

Xe tự hành Curiosity Marsbot chạy trên sao Hỏa, được vận hành từ xa bởi các lập trình viên trên Trái Đất. Bằng cách gửi đi các mã điều khiển từ một bàn phím ma trận, lập trình viên điều khiển quá trình di chuyển của Marbot như sau:

Mã điều khiển	**Ý nghĩa**
1b4	Marsbot bắt đầu chuyển động
c68	Marsbot đứng im
444	Rẽ trái 90 độ so với phương chuyển động gần nhất
666	Rẽ phải 90 độ so với phương chuyển động gần nhất
dad	Bắt đầu để lại vết trên đường
cbc	Chấm dứt để lại vết trên đường
999	Tự động đi theo lộ trình ngược lại. Không vẽ vết, không nhận mã khác cho tới khi kết thúc lộ trình ngược. Mô tả: Marsbot được lập trình để nhớ lại toàn bộ lịch sử các mã điều khiển và khoảng thời gian giữa các lần đổi mã. Vì vậy, nó có thể đảo ngược lại lộ trình để quay về điểm xuất phát.

Sau khi nhận mã điều khiển, Curiosity Marsbot sẽ không xử lý ngay, mà phải đợi lệnh kích hoạt mã từ bàn phím Keyboard & Display MMIO Simulator. Có 3 lệnh như vậy:

Kích hoạt mã	**Ý nghĩa**
Phím Enter	Kết thúc nhập mã và yêu cầu Marsbot thực thi
Phím Delete	Xóa toàn bộ mã điều khiển đang nhập.
Phím Space	Lặp lại lệnh đã thực hiện trước đó.

Hãy lập trình để Marsbot có thể hoạt động như đã mô tả. Đồng thời bổ sung thêm tính năng: mỗi khi gửi một mã điều khiển cho Marsbot, hiển thị mã đó lên màn hình console để người xem có thể giám sát lộ trình của xe.

1.2 Phân tích cách làm

1.2.1 Xử lý đầu vào

Đầu vào có 2 phần:

1. Bàn phím ma trận Digital Lab
2. Bàn phím Keyboard & Display MMIO

Để nhận đầu vào từ bàn phím ma trận, ta sẽ sử dụng ngắt
Để nhận đầu vào từ bàn phím Keyboard MMIO, ta sẽ sử dụng polling

1.2.2 Logic chương trình

Thực thi các mã lệnh

Giữ một cấu trúc dữ liệu **Stack** toàn cục phục vụ cho chức năng **999**, ý tưởng là chứa xen kẽ thời gian giữa các câu lệnh vào câu lệnh
Giữ 2 biến **LOW_TIME** và **HIGH_TIME** để chứa thời gian khi câu lệnh trước bắt đầu thực thi (sử dụng **SYSCALL** 30)
Giữ một biến toàn cục **CONTROL_CODE** chứa câu lệnh người dùng đang nhập
Giữ một biến toàn cục **MARS_CURRENT_HEAD** để chứa hướng quay hiện tại của MARSBOT (Các câu lệnh rotate sẽ thay đổi biến này)
Lưu ý: Nếu MARSBOT đang di chuyển thì sẽ không chấp nhận mã di chuyển, nếu MARSBOT đang dừng cũng sẽ không chấp nhận mã dừng

Khi người dùng nhập xong lệnh ở bàn phím ma trận và ấn phím trên bàn phím MMIO hàm xử lý bàn phím MMIO sẽ chạy với mã giả sau:

[H] **Algorithm 1** Xử lý phím MMIO

```
1:  function process_key(KEY)
2:      if KEY = Enter then
3:          if CONTROL_CODE = 999 then
4:              Tính thời gian hiện tại (SYSCALL 30)
5:              Tính thời gian giữa câu lệnh trước và câu lệnh hiện tại
6:              Đẩy khoảng thời gian lên STACK
7:              Chạy hàm xử lý 999
8:          else if CONTROL_CODE ≠ 999 then
9:              Chạy hàm xử lý CONTROL_CODE
10:             if Xử lý thành công then
11:                 Tính thời gian hiện tại (SYSCALL 30)
12:                 Tính thời gian giữa câu lệnh trước và câu lệnh hiện tại
13:                 Đẩy khoảng thời gian lên STACK
14:                 Đẩy CONTROL_CODE lên stack
15:             end if
16:             Xóa nội dung CONTROL_CODE
17:         end if
18:     else if KEY = Delete then
19:         Xóa nội dung CONTROL_CODE
20:     else if KEY = Space then
```

```
21:         Xem câu lệnh ở đầu STACK
22:         Chạy hàm xử lý CONTROL_CODE
23:         if Xử lý thành công then
24:             Tính thời gian hiện tại (SYSCALL 30)
25:             Tính khoảng thời gian giữa câu lệnh trước và câu lệnh hiện tại
26:             Đẩy khoảng thời gian lên STACK
27:             Đẩy CONTROL_CODE lên stack
28:         end if
29:     end if
30: end function
```

Hàm xử lý câu lệnh sẽ thực thi nếu lệnh hợp lệ có nguyên mẫu sau, cài đặt của hàm dưới trong file *main.asm*:

Algorithm 2 Xử lý câu lệnh: bot_exec

Đầu vào: *CONTROL_CODE*
Đầu ra: *BOOLEAN* 0 nếu thành công, -1 nếu thất bại

Mã giả sau đây mô tả cách hoạt động của hàm Xử lý lệnh 999:

[H] **Algorithm 3** Xử lý lệnh 999

```
1: function back_track(void)
2:     Tắt Tracking
3:     T ← STACK.TOP                                   ▷ Lấy thời gian
4:     STACK.POP()                                     ▷ Pop thời gian
5:     C ← STACK.TOP                                   ▷ Thử lấy câu lệnh
6:     if Lấy Top thất bại then
7:         return void
8:     end if
9:     if C ≠ STOP then
10:        STACK.PUSH(T)                               ▷ Đẩy thời gian lên Stack
11:        STACK.PUSH(STOP)                            ▷ Đẩy câu lệnh STOP lên Stack
12:    end if
13:    while STACK not Empty do
14:        T ← STACK.TOP                               ▷ Lấy thời gian
15:        STACK.POP()                                 ▷ Pop thời gian
16:        C ← STACK.TOP                               ▷ Lấy câu lệnh
17:        STACK.POP()                                 ▷ Pop câu lệnh
18:        if C = STOP then
19:            M ← 1                                   ▷ Biến Moving = 1
20:            Quay 180 độ
21:            Di chuyển trong T mili giây
22:            Ngừng di chuyển
23:        else if C = GO then
24:            M ← 0                                   ▷ Biến Moving = 0
25:            Quay 180 độ
26:            Ngừng di chuyển
27:        else if C = TURN_LEFT_90 then
28:            Quay phải 90 độ
29:            if M = 1 then
30:                Di chuyển trong T mili giây
31:                Ngừng di chuyển
32:            else if M = 0 then
33:                Ngừng di chuyển
34:            end if
35:        else if C = TURN_RIGHT_90 then
36:            Quay trái 90 độ
37:            if M = 1 then
38:                Di chuyển trong T mili giây
39:                Ngừng di chuyển
40:            else if M = 0 then
41:                Ngừng di chuyển
42:            end if
43:        else if C = TRACK_ON then
44:            if M = 1 then
45:                Di chuyển trong T mili giây
46:                Ngừng di chuyển
47:            else if M = 0 then
```

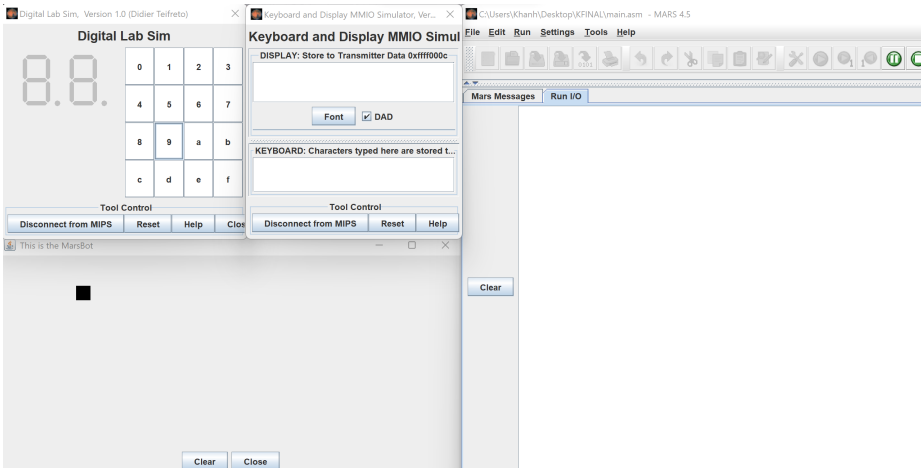
```
            Ngừng di chuyển
        end if
    else if C = TRACK_OFF then
        if M = 1 then
            Di chuyển trong T mili giây
            Ngừng di chuyển
        else if M = 0 then
            Ngừng di chuyển
        end if
    end if
    Lấy thời gian hiện tại
    Lưu vào LOW_TIME và HIGH_TIME
  end while
end function
```

1.3 Các hàm phụ

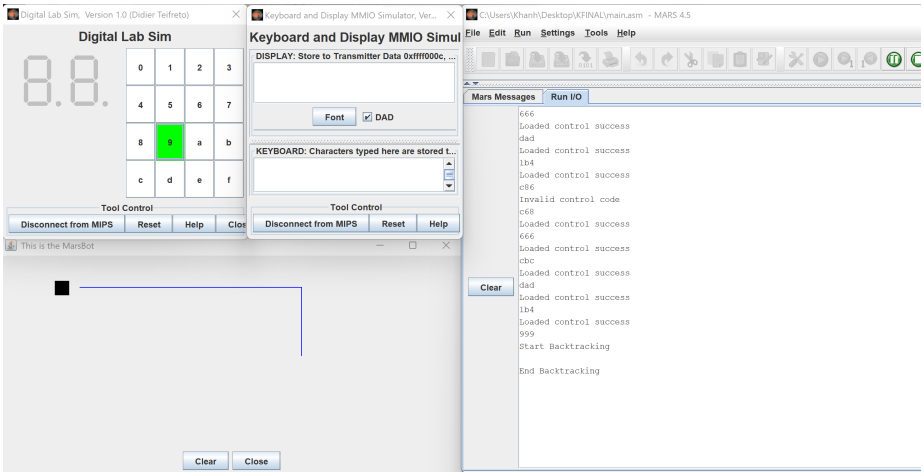
1. **sub64** Trừ số 64 bit
2. **str_append** Gán kí tự vào xâu
3. **strlen** Tính độ dài của xâu

Cài đặt của CTDL Stack trong file *stack.asm*
Các hàm tương tác với Marsbot trong file *marsbot.asm*
Cài đặt ngắt trong file *main.asm*
Lưu ý: Vì hàm sleep có thể không chính xác nên Marsbot có thể lệch với vị trí ban đầu khi đi ngược lại

1.4 Kết quả chạy chương trình



Hình 1.1: Trước khi chạy backtrack



Hình 1.2: Sau khi chạy backtrack

Bài 2
Biểu thức trung tố hậu tố

2.1 Đề bài

Viết chương trình tính giá trị biểu thức bất kỳ bằng phương pháp duyệt biểu thức hậu tố.
Các yêu cầu cụ thể:

1. Nhập vào biểu thức trung tố, ví dụ: 9 + 2 + 8 * 6
2. In ra biểu thức ở dạng hậu tố, ví dụ: 9 2 + 8 6 * +
3. In ra biểu thức ở dạng hậu tố, ví dụ: 9 2 + 8 6 * +

Toán tử bao gồm các phép toán cộng, trừ, nhân, chia lấy thương (/), chia lấy dư (%), đóng mở ngoặc
Các số từ 0 - 99

2.2 Phân tích cách làm

Chương trình sẽ có 2 bước chính:

1. Chuyển đổi từ trung tố sang hậu tố
2. Tính giá trị của biểu thức hậu tố

2.2.1 Chuyển đổi từ trung tố sang hậu tố

Các cấu trúc dữ liệu cần thiết:

- Stack
- Queue

Giả dụ đầu vào không chứa token ngoại lệ. Thuật toán chuyển đổi từ trung tố sang hậu tố như sau:

[H] **Algorithm 4** INFIX TO POSTFIX

Đầu vào: INFIX
Đầu ra: POSTFIX

while $INFIX$ còn token **do**
  $X \leftarrow Token$
  if X là số **then**
    $QUEUE.ENQUEUE(X)$
  else if X là toán tử **then**
    while $STACK.TOP()$ ưu tiên hơn hoặc bằng X **do**
      $QUEUE.ENQUEUE(STACK.TOP())$
      $STACK.POP()$
    end while
    $STACK.PUSH(X)$
  else if $X = ($ **then**
    $STACK.PUSH(X)$
  else if $X =)$ **then**
    while $STACK.TOP() \neq ($ **do**
      $QUEUE.ENQUEUE(STACK.TOP())$
      $STACK.POP()$
    end while
    $STACK.POP()$
  end if
end while
while $STACK$ not Empty **do**
  $X \leftarrow STACK.TOP$
  $STACK.POP()$
  $QUEUE.ENQUEUE(X)$
end while

Nếu đầu vào không hợp lệ ta cần xử lý 3 trường hợp sau:

1. Đầu vào chứa kí tự không hợp lệ
2. Đầu vào thừa ngoặc trái '('
3. Đầu vào thừa ngoặc phải ')'

Thuật toán chuyển đổi từ hậu tố sang trung tố xử lý ngoại lệ như sau:

[H] **Algorithm 5** INFIX TO POSTFIX

Đầu vào: INFIX
Đầu ra: POSTFIX

while $INFIX$ còn token **do**
  $X \leftarrow Token$
  if X là số **then**
    $QUEUE.ENQUEUE(X)$
  else if X là toán tử **then**
    while $STACK.TOP()$ ưu tiên hơn hoặc bằng X **do**
      $QUEUE.ENQUEUE(STACK.TOP())$
      $STACK.POP()$
    end while
    $STACK.PUSH(X)$
  else if $X = ($ **then**
    $STACK.PUSH(X)$
  else if $X =)$ **then**
    while $STACK.TOP() \neq ($ và $STACK$ NOT EMPTY **do**
      $QUEUE.ENQUEUE(STACK.TOP())$
      $STACK.POP()$
    end while
    if $STACK.TOP() \neq ($ **then**
      Đầu vào thừa dấu ')'
    end if
    $STACK.POP()$
  else
    Đầu vào chứa kí tự không hợp lệ
  end if
end while
while $STACK$ NOT EMPTY **do**
  $X \leftarrow STACK.TOP$
  if $X = ($ **then**
    Đầu vào thừa dấu '('
  end if
  $STACK.POP()$
  $QUEUE.ENQUEUE(X)$
end while

Áp dụng thuật toán trên để cài đặt hàm *infix_to_postfix* trong file *main.asm*

2.2.2 Tính biểu thức hậu tố

Cấu trúc dữ liệu cần thiết: Stack Giả dụ đầu vào là một biểu thức hậu tố hợp lệ, thuật toán tính biểu thức hậu tố như sau:

[H] **Algorithm 6** EVALUATE POSTFIX

Đầu vào: POSTFIX
Đầu ra: RESULT
1: **while** $POSTFIX$ còn token **do**
2: $X \leftarrow token$
3: **if** X là số **then**
4: STACK.PUSH(X)
5: **else if** X là toán tử **then**
6: $A \leftarrow STACK.TOP()$
7: $STACK.POP()$
8: $B \leftarrow STACK.TOP()$
9: $STACK.POP()$
10: $C \leftarrow X(A, B)$
11: $STACK.PUSH(C)$
12: **end if**
13: **end while**
14: $RESULT \leftarrow STACK.TOP()$
15: **return** $RESULT$

Áp dụng thuật toán trên để cài đặt hàm *evaluate_postfix* trong file *main.asm*

2.3 Các hàm phụ

1. **strlen** Tính độ dài của xâu
2. **substrcpy** Copy xâu con của string
3. **remove_white_spaces** Loại bỏ các kí tự Space khỏi string
4. **binary_op** Thực hiện toán tử lên 2 toán hạng

Các hàm trên giúp cài đặt thuật toán, mã nguồn của chúng ở trong file *main.asm* và *helper.asm*
Cài đặt của CTDL Stack và Queue lần lượt ở trong file mã nguồn *Stack.asm* và *Queue.asm*

2.4 Kết quả chạy chương trình

```
                INFIX TO POSTFIX
1. Convert infix to postfix and evaluate
2. Exit
Enter your choice: 1

Input infix: 3 + 2 * 4 - 8 % 3 / (3 - 4)
Equivalent postfix: 3 2 4 * + 8 3 % 3 4 - / -
Result: 13
```

Hình 2.1: Biểu thức hợp lệ

```
                INFIX TO POSTFIX
1. Convert infix to postfix and evaluate
2. Exit
Enter your choice: 1

Input infix: ((4+3)
Input error
```

Hình 2.2: Biểu thức không hợp lệ