TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN TPHCM
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO MÔN HỌC
KIẾN TRÚC MÁY TÍNH VÀ HỢP NGỮ

Đồ án 2

THƯ VIỆN TIME

Nhóm sinh viên thực hiện:	Vũ Lê Thế Anh	1612838
	Nguyễn Lê Hồng Hạnh	1612849
	Phan Quốc Tuấn	1612902

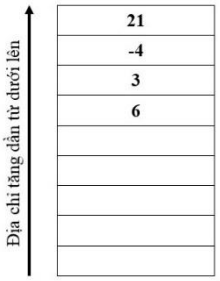
Mục lục

1. Quy tắc viết và gọi hàm: 3
1.1. Quy tắc viết hàm: 4
1.2. Quy tắc gọi hàm: 4
2. Cách thức cài đặt các hàm quan trọng: 5
2.1. Nhập dữ liệu 5
2.2. Nhập ngày, tháng, năm và kiểm tra tính hợp lệ: 5
2.3. Xuất chuỗi TIME dưới dạng DD/MM/YYYY: 6
2.4. Chuyển đổi giữa các định dạng: 6
2.5. Các hàm lấy ngày, tháng, năm từ chuỗi: 7
2.6. Kiểm tra năm nhuận: 7
2.7. Khoảng cách giữa hai chuỗi thời gian: 8
2.8. Xác định ngày trong tuần của ngày vừa nhập: 9
2.9. Tìm hai năm nhuận kề sau: 10

1. Quy tắc viết và gọi hàm:

Trong MIPS không thực sự tồn tại khái niệm "hàm" mà chỉ đơn thuần là sử dụng một nhãn dán (label) để xác định vị trí của các dòng lệnh liên tục tiếp theo, đóng vai trò tương ứng như một hàm. Các thanh ghi của MIPS cũng không thể sử dụng như một biến cục bộ mà hoàn toàn mang ý nghĩa toàn cục, tức là nếu ở bất kì dòng lệnh nào ta thay đổi giá trị của thanh ghi thì sự thay đổi đó là vĩnh viễn, chỉ có thể khôi phục lại nếu được lưu giữ trước đó, chứ không tự động giữ nguyên không thay đổi giá trị như khi chúng ta cài đặt trong các ngôn ngữ lập trình như C/C++. Điều đó đặt ra sự cần thiết phải có một quy tắc khi viết và gọi hàm để đảm bảo các thanh ghi hoạt động trơn tru như mong muốn mà không bị ghi đè lên nhau.

Quy tắc này xoay quanh stack. *Stack* là một vùng nhớ trên máy tính hoạt động trên nguyên tắc Last in First out (vào sau ra trước). Các dữ liệu quan trọng trong thanh ghi sẽ được lưu trữ tại đây theo một quy ước chung để thuận tiện cho việc kiểm tra, và ghép giữa nhiều đoạn lập trình của nhiều người khác nhau. Có hai hành động thường thực hiện trên stack là push (đẩy vào) để đưa dữ liệu vào stack và pop (bật ra) để lấy dữ liệu ở trên cùng stack.



Hình 1. Minh họa về stack

Nhắc lại, các thanh ghi hoàn toàn là như nhau theo nghĩa toàn cục, tuy nhiên người ta vẫn chia nó ra thành nhiều loại khác nhau theo quy ước để thống nhất. Một trong số các loại đó là:

Bảng 1. Bảng một số thanh ghi quan trọng

Thanh ghi	Số hiệu	Chức năng	Trách nhiệm lưu trữ
$zero	0	Số 0	
$v0 - $v1	2 – 3	Giá trị trả về	Người gọi
$a0 - $a3	4 – 7	Tham số hàm	Người viết
$t0 - $t7	8 – 15	Thanh ghi tạm (temporary)	Người gọi
$s0 - $s7	16 – 23	Tham ghi lưu (saved)	Người viết
$sp	29	Địa chỉ đầu stack	
$ra	31	Địa chỉ trở về	Người viết

Ở đây, các thanh ghi *$s, $a, $ra* là những thanh ghi mà người viết hàm phải đảm bảo không thay đổi, tức là đầu hàm phải được lưu trữ lại trong stack trước khi thực hiện bất cứ lệnh gì. Ngược lại, các thanh ghi *$t, $v* là những thanh ghi tạm thời, người viết hàm không có trách nhiệm lưu trữ mà trách nhiệm này nằm ở người gọi hàm nếu họ không muốn sau khi gọi hàm thanh ghi này bị ghi đè.

Đa phần trường hợp, các thanh ghi *$s* được dùng như một biến cố định, để khi hoạt động hàm lồng hàm cũng không bị ảnh hưởng. Thanh ghi *$t* được dùng như một biến tạm thời, cục bộ ở một đoạn ngắn, không phải lo lắng khi gọi hàm nó bị thay đổi (có thể là do không còn dùng tới nữa từ trước khi gọi hàm).

Do đó, khi viết và gọi hàm ta có một số quy tắc:

1.1. Quy tắc viết hàm:

- Đầu hàm phải push toàn bộ các thanh ghi $ra, $a, $s vào stack để lưu trữ (có thể chỉ cần push những thanh ghi $s mình sử dụng trong hàm)
- Viết các câu lệnh thỏa yêu cầu của hàm bình thường, các tham số truyền vào ($a) có thể lấy ra từ địa chỉ tương ứng của nó trong stack, các thanh ghi $s được sử dụng thoải mái (do đã lưu lại từ trước)
- Cuối hàm phải pop toàn bộ các thanh ghi đã lưu và trả stack về trạng thái ban đầu

Một khung hàm ví dụ là như sau:

```
addi    $sp, $sp, -12   # Chuẩn bị 12 = 3 * 4 byte trong stack
sw      $ra, 8($sp)     # Đưa 3 thanh ghi vào stack
sw      $a0, 4($sp)
sw      $s0, 0($sp)
…
# Các dòng lệnh hàm, lưu giá trị trả về (nếu có) trên các thanh ghi $v
…
lw      $s0, 0($sp)     # Hồi phục giá trị 3 thanh ghi ban đầu
lw      $a0, 4($sp)
lw      $ra, 8($sp)
addi    $sp, $sp, 12    # Trả stack về trạng thái ban đầu
jr      $ra             # Trả về
```

Có thể tưởng tượng một vấn đề dễ mắc phải là khi gọi hàm lồng hàm, điển hình nhất là thanh ghi $ra sẽ bị thay đổi, nếu không lưu trữ lại trước thì ở dòng lệnh cuối sẽ không còn là địa chỉ ban đầu nữa.

1.2. Quy tắc gọi hàm:

- Trước khi gọi hàm phải lưu trữ bất kì thanh ghi $t và $v mà không muốn bị đổi giá trị sau khi chạy hàm (thường ít thấy do thường đây là các thanh ghi chỉ dùng tạm thời, cục bộ)
- Lưu các tham số truyền vào hàm vào các thanh ghi $a
- Nhảy đến nhãn dán của hàm và lưu điểm trở về vào thanh ghi $ra

Một khung gọi hàm ví dụ:

```
…
addi    $sp, $sp, -4    # Lưu thanh $t0 vào stack
sw      $t0, 0($sp)
add     $a0, $s0, $0    # Truyền vào hàm giá trị trong thanh $s0
jal     MyFunction      # Gọi hàm MyFunction($s0)
lw      $t0, 0($sp)
```

```
addi    $sp, $sp, 4      # Trả stack về bình thường
…
```

Sự cần thiết phải lưu lại thanh ghi $t là do trong hàm chỉ đảm bảo không đổi các thanh $ra, $s, $a chứ không đảm bảo cho thanh $t. Thậm chí trong hàm còn thường xuyên dùng thanh $t làm thanh ghi dữ liệu cục bộ cho riêng nó, nên việc ghi đè là khả dĩ.

2. Cách thức cài đặt các hàm quan trọng:

2.1. Nhập dữ liệu

Quy trình nhận input từ người dùng gồm các bước:

- Người dùng nhập vào một chuỗi các ký tự
- Chuỗi ký tự này được lưu vào một chuỗi buffer tạm
- Kiểm tra tính hợp lệ của chuỗi ký tự:
    - Nếu ngữ cảnh là nhập vào một số nguyên dương, kiểm tra từng ký tự một xem có tồn tại ký tự nào ngoài khoảng [‘0’, ‘9’]
    - Nếu ngữ cảnh là một lựa chọn (ví dụ, ‘A’, ‘B’ hoặc ‘C’), kiểm tra xem đó có là một trong các lựa chọn cho phép
- Nếu hợp lệ, chuỗi sẽ được nhập vào và chuyển về đúng dạng của nó:
    - Nếu là số nguyên, chuyển về biến dạng số nguyên (word)
    - Nếu là ký tự/chuỗi, lưu vào mảng ký tự phù hợp
- Nếu không hợp lệ, lặp lại từ bước yêu cầu nhập xuất

Cụ thể, ở phần chuyển chuỗi về số nguyên, sau khi đã kiểm tra phù hợp ngữ cảnh (không có ký tự không hợp lệ), lần lượt thực hiện:

- Đặt biến số nguyên (word) trả về là s, ban đầu $s = 0$
- Duyệt mảng ký tự từ trái sang, cập nhật $s = 10s + c - '0'$ (giá trị $'0' = 48$ nhằm chuyển từ ký tự thành số nguyên)
- Trả về s

Trong trường hợp đồ án này, không nhận các số có giá trị quá 10^9 (nếu nhập quá sẽ báo lỗi).

Hàm GetNum thực hiện công việc này, tương ứng trong C/C++ của nó là:

int GetNum()

Hàm không nhận vào tham số, yêu cầu nhập chuỗi được thực hiện nhờ opt code 4 của syscall:

```
la $a0, Buffer
li  $v0, 4
syscall
```

2.2. Nhập ngày, tháng, năm và kiểm tra tính hợp lệ:

Sử dụng hàm GetNum ở trên để yêu cầu người dùng nhập vào lần lượt ngày, tháng và năm. Thực hiện kiểm tra sau khi đã có đủ 3 tham số bằng hàm CheckDate, tương ứng C/C++ là:

int CheckDate(int day, int month, int year)

Hàm nhận vào 3 tham số ngày, tháng, năm và trả về 1 nếu đó là ngày hợp lệ và 0 nếu không hợp lệ.

Việc kiểm tra gồm các bước:

- Kiểm tra năm, theo yêu cầu, $1900 \leq y \leq 9999$ (do chương trình chỉ xét đến các năm có 4 chữ số). Ngoài ra, ở đây cần kiểm tra thêm thông tin năm nhuận hay không (quy trình được trình bày ở phần sau)
- Kiểm tra tháng, $1 \leq m \leq 12$
- Để kiểm tra ngày, trước hết phải có một mảng số nguyên chứa số lượng ngày trong mỗi tháng

  int MonthDayCount[13] = { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 }

  Ở đây không xét tháng 0 và tháng 2 là ngày mặc định không phải năm nhuận (28 ngày)
- Kiểm tra ngày: $1 \leq d \leq MonthDayCount[m]$, trong trường hợp năm nhuận và $m = 2$ (tháng đang xét là tháng 2), kiểm tra với $d' = d - 1$

Nếu có sự bất hợp lý trong thông tin nhập vào, quay trở lại yêu cầu người dùng nhập lại.

2.3. Xuất chuỗi TIME dưới dạng DD/MM/YYYY:

Tương ứng C/C++:

char* Date(int day, int month, int year, char* TIME)

Hàm nhận vào 4 tham số là ngày, tháng, năm và địa chỉ của chuỗi kết quả. Hàm trả về địa chỉ chuỗi kết quả.

Đối với ngày và tháng, ta thực hiện phép chia cho 10 và lấy phần thương cho số ở hàng chục và phần dư cho số ở hàng đơn vị.

Đối với năm, lần lượt thực hiện phép chia cho 10 và điền số dư từ phải sang cho tới đủ hết 4 ký số.

Các ký số được cộng với $48 = '0'$ nhằm chuyển thành ký tự và lưu vào mảng. Các ký tự '/' được thêm vào tại vị trí phù hợp (sau mỗi bước thêm ngày và tháng).

2.4. Chuyển đổi giữa các định dạng:

Hàm tương ứng trong C/C++:

char* Convert(char* TIME, char type)

Hàm nhận vào TIME là địa chỉ của chuỗi cần chuyển đổi và type là kiểu biểu diễn sau khi chuyển đổi. Hàm trả về địa chỉ của TIME, tức hàm sẽ thực hiện chuyển đổi ngay trên chuỗi TIME nhận vào.

+ Định dạng A: Lưu các ký tự của DD vào 2 biến tạm, chuyển MM vào vị trí của DD (dùng lb và sb) rồi chuyển các ký tự đã lưu vào vị trí tương ứng của MM

+ Định dạng B và C:

Tạo một mảng các chuỗi (asciiz) cho biết tên của từng tháng:

char* MonthName[12] = { January, February, March, April, May, June, July, August, September, October, November, December }

Lần lượt lưu các ký tự của DD và YYYY vào các biến tạm.

Sử dụng hàm Month để tìm ra tháng dưới dạng số nguyên m.

Chuyển lần lượt từng ký tự của MonthName[m – 1] vào vị trí tương ứng.

Chuyển lần lượt các ký tự đã lưu của DD và YYYY vào vị trí tương ứng.

Trong quá trình, thêm khoảng trắng và dấu phẩy ở vị trí tương ứng.

2.5. Các hàm lấy ngày, tháng, năm từ chuỗi:

Các hàm tương ứng trong C/C++:

int Day(char* TIME)

int Month(char* TIME)

int Year(char* TIME)

Các hàm đều nhận vào TIME là địa chỉ chuỗi thời gian và trả về con số là ngày, tháng hoặc năm tương ứng.

Làm ngược lại với quá trình nhập chuỗi, ta có được ngày, tháng, năm từ chuỗi nhập vào.

Đối với ngày và tháng, nếu đánh số thứ tự cho chuỗi từ trái qua, bắt đầu từ 0, ta có kết quả:

$$n = 10(s[1] - '0') + s[0] - '0'$$

Nhắc lại phép trừ $'0'$ nhằm mục đích chuyển ký tự thành ký số.

Đối với năm, chuyển đổi chuỗi thành số tương tự như cách làm trong hàm nhập GetNum (chạy vòng lặp với $n_0 = 0, n_i = 10n_{i-1} + s[i]$).

2.6. Kiểm tra năm nhuận:

Hàm tương ứng trong C/C++:

int LeapYear(char* TIME)

Hàm nhận vào TIME là địa chỉ chuỗi cần kiểm tra và trả về 1 nếu là năm nhuận, 0 nếu ngược lại.

Đầu tiên sử dụng hàm Year để rút năm ra khỏi chuỗi TIME, thực hiện kiểm tra trên năm y này.

Nhắc lại định nghĩa năm nhuận: năm chia hết cho 4, nhưng nếu chia hết cho cả 100 thì phải chia hết cho 400. Nói cách khác, năm nhuận là năm chia hết cho 400 hoặc chia hết cho 4 và không chia hết cho 100.

Biểu thức kiểm tra: y % 400 == 0 || (y % 4 == 0 && y % 100 != 0)

Sơ đồ kiểm tra:

If (y % 100 == 0):

If (y % 400 == 0): năm nhuận (chia hết cho 100 và 400)

Else: năm không nhuận (chia hết cho 100 nhưng không hết cho 400)

Else:

If (y % 4 == 0): năm nhuận (chia hết cho 4 nhưng không hết cho 100)

Else: năm không nhuận (không chia hết cho 4 hay 100)

2.7. Khoảng cách giữa hai chuỗi thời gian:

Hàm tương ứng trong C/C++:

int GetTime(char* TIME_1, char* TIME_2)

Để thực hiện phép tính, cần thêm các hàm bổ trợ:

+ Hàm tính số ngày từ ngày đó đến ngày đầu tiên của năm tương ứng:

int DaysFromYearStart(char* TIME)

Hàm nhận vào một chuỗi thời gian và trả về số ngày từ ngày mà chuỗi biểu diễn đến ngày 1/1 của năm đó.

Đầu tiên kiểm tra năm nhuận, nếu thật là năm nhuận và $m > 2$ thì biến trả về sẽ khởi đầu là $n = 1$, ngược lại biến trả về sẽ khởi đầu là $n = 0$.

Sử dụng mảng số ngày trong tháng như trình bày ở phần kiểm tra tính hợp lệ, chạy vòng lặp cộng tất cả các ngày trong tháng của các tháng từ 1 đến $m - 1$ (nếu $m = 1$, không cộng gì).

Cộng $d - 1$ vào kết quả, ta có số ngày cần tìm.

Công thức:

$$n = LeapYear(y) + \sum_{i=1}^{m-1} MonthDayCount[i] + (d - 1)$$

+ Hàm tính số ngày giữa hai năm:

int DaysBetweenYearStarts(int year1, int year2)

Hàm nhận vào hai số nguyên là hai năm cần kiểm tra ($year1 \leq year2$) và trả về số ngày giữa hai năm.

Ta thực hiện vòng lặp từ $year1$ đến $year2 - 1$, lần lượt cộng thêm số ngày của năm: 365 nếu là năm không nhuận và 366 nếu là năm nhuận.

Công thức:

$$n = \sum_{i=year1}^{year2-1} [LeapYear(i) + 365]$$

+ Hàm kiểm tra TIME_1 trước hay sau TIME_2:

int IsBefore(int time1, int year1, int time2, int year2)

Hàm nhận vào mỗi thời điểm là thời gian từ ngày đó đến ngày đầu năm (time) và năm của thời điểm đó (year). Hàm trả về 1 nếu TIME_1 trước TIME_2, trả về 0 nếu ngược lại.

Để kiểm tra, trước hết xem năm rồi sau đó nếu trong cùng năm ta mới xét đến ngày từ đó đến đầu năm.

Biểu thức cần kiểm tra: (year1 < year2) || (year1 == year2 && time1 < time2)

Với các hàm phụ trợ trên, ta tìm số ngày giữa hai thời điểm bằng công thức:

$$GetTime(TIME\_1, TIME\_2) = DaysBetweenYearStarts\big(Year(TIME\_1), Year(TIME\_2)\big) + DaysFromYearStart(TIME\_2) - DaysFromYearStart(TIME\_1)$$

Trước khi thực hiện phép tính, cần kiểm tra xem nếu TIME_1 sau TIME_2 thì đổi ngược 2 thời gian lại.

2.8. Xác định ngày trong tuần của ngày vừa nhập:

Hàm tương ứng trong C/C++:

char* Weekday(char* TIME)

Đầu tiên, chuẩn bị một mảng chuỗi trả về tương ứng ký hiệu từng ngày trong tuần:

char* WeekDayName[7] = { Mon, Tues, Wed, Thurs, Fri, Sat, Sun }

Để tìm ngày trong tuần, ta chọn một ngày làm mốc, chọn một ngày thứ Hai bất kì, giả sử chọn ngày 02/04/2018.

Sử dụng hàm GetTime ta có được số ngày khoảng cách giữa ngày cần xác định thứ và ngày làm mốc. Từ đó có:

- Nếu ngày cần xác định sau ngày làm mốc, tìm $t = GetTime(DATE, AnchorDate)$ % 7
- Nếu trước, tìm $t = 7 - GetTime(DATE, AnchorDate)$ % 7

Ngày trong tuần trả về sẽ là $WeekDayName[t]$.

2.9. Tìm hai năm nhuận kề sau:

Hàm tương ứng trong C/C++:

pair<int,int> NextTwoLeapYears(char* TIME)

Hàm nhận vào chuỗi thời gian và trả về hai con số ứng với hai năm nhuận kề sau đó.

Đầu tiên, ta trích năm ra khỏi chuỗi thời gian (bằng hàm Year) rồi tìm $y' = y - y \% 4$. Phép tính này sẽ cho ta y' chính là năm chia hết cho 4 gần nhất trước hoặc bằng năm y.

Kiểm tra các năm $y' + 4$, $y' + 8$ và $y' + 12$ và chọn ra 2 số bé nhất để trả về.

Thực hiện theo sơ đồ như sau, lưu ý là chỉ có tối đa 1 trong 3 năm là không nhuận do chia hết 100 mà không chia hết 400:

- Kiểm tra $y' + 4$, nếu đây là năm nhuận, tiếp tục hàm (coi như đã tìm thấy 1 năm). Nếu không, kết thúc hàm và trả về $y' + 8$ và $y' + 12$.
- Kiểm tra $y' + 8$, nếu đây là năm nhuận thì kết thúc hàm và trả về $y' + 4$ và $y' + 8$. Nếu không phải năm nhuận thì trả về $y' + 4$ và $y' + 12$.

Do yêu cầu sử dụng hàm cài đặt kiểm tra năm nhuận trên chuỗi nên ở các bước kiểm tra, ta tạo chuỗi bằng hàm Date rồi dùng hàm LeapYear kiểm tra trên chuỗi đó. Cần sử dụng biến tạm để lưu trữ, tránh trường hợp ghi đè lên chuỗi nhập vào. Ngoài ra cũng cần chú ý trường hợp $y' = 9992$ và $y' = 9996$ do khi tính sẽ tạo thành năm có hơn 4 chữ số (thực tế không bị lỗi do chuỗi tạo thành có năm 0000 và/hoặc 0004 là các năm nhuận nên "vô tình" vẫn chấp nhận).