Universidade Federal Fluminense – UFF
Escola de Engenharia – TCE
Curso de Engenharia de Telecomunicações – TGT
Programa de Educação Tutorial – PET

Projetos PET-Tele

Geração dinâmica de *webpage* baseada no acesso via Internet de banco de dados implementado com Lua

(Versão: A2013M06D20)

Autores: Bruno Martins Costa

Tutor: Alexandre Santos de la Vega

Niterói – RJ
Setembro / 2012

1 Introdução

Lua é uma linguagem de programação, interpretada e com tipagem dinâmica de variáveis, utilizada para *scripting*, desenvolvida no Laboratório Tecgraf, da PUC-RJ. Ela é, hoje, a linguagem de *scripting* mais usada em jogos computacionais (*games*). A sua utilização em *websites* é interessante devido à sua simplicidade (sendo assim de fácil aprendizado), além de possuir uma vasta biblioteca de funções para o manuseio de *strings* e arquivos.

A estrutura de dados fundamental em Lua é a tabela, implementada na forma de uma tabela associativa, onde cada registro é formado por uma chave de acesso e um valor. Esta possui grande flexibilidade, podendo representar vários tipos teóricos de estruturas de dados: vetores, listas, filas, tabelas de símbolos, conjuntos, gráficos, registro e *hash*. Ela permite que possamos organizar, relacionar e hierarquizar dados, possibilitando, por exemplo, a criação de um banco de dados, que pode ser acessado por um programa escrito em Lua.

Uma das formas de interagir com um base de dados estruturada em tabelas de Lua é através do comando *dofile*, que permite que trechos de código armazenados em arquivos possam ser carregados dinamicamente para um programa Lua durante a execução do mesmo. Assim, pode-se ter a base de dados inserida em tabelas armazenadas em um simples arquivo Lua, as quais podem ser carregadas dinamicamente pelo programa.

Com o uso da linguagem de *binding* CGILua, é possível criar *webpages* dinâmicas através de código Lua embarcado no código HTML.

Utilizando as ferramentas mencionadas acima, podemos criar um banco de dados simples, usando tabelas em Lua, acessá-lo através do *dofile*, processar os dados através de um programa Lua embarcado em uma página HTML e atualizar uma *webpage* com os resultados.

Nossa proposta, através deste trabalho, é implementar tal mecanismo, verificando suas vantagens e desvantagens em relação às ferramentas mais utilizadas no desenvolvimento *Web* para a manipulação de banco de dados.

2 Motivação

No *website* do Grupo PET-Tele (`http://www.telecom.uff.br/pet`), na Seção *Integrantes*, há uma série de informações acerca de todos os integrantes e colaboradores que já passaram pelo grupo, tais como: datas de ingresso e egresso, aniversários, local de trabalho atual, entre outros. Várias dessas informações se repetem em algumas páginas do *website*. A cada modificação no quadro de integrantes do grupo, todas as páginas envolvidas precisam ser atualizadas manualmente. Surgiu, então, a ideia de automatizar este processo. Assim, as páginas envolvidas poderiam ser carregadas dinamicamente, sendo as informações geradas a partir do acesso a uma pequena base de dados. Isto poderia ser feito usando uma combinação PHP-MySQL. Porém, como este é um mecanismo que o grupo já domina, decidiu-se usar a associação HTML-CGILua-Lua, acrescentando novos conhecimentos ao grupo. Além disso, acreditamos que uma base de dados codificada em um arquivo de texto comum é mais fácil de portar do que uma codificada com MySQL.

A linguagem de programação Lua foi inserida no Grupo PET-Tele devido à sua utilização no *Middleware* Aberto do Sistema Nipo-Brasileiro de Televisão Digital (ISDB-TB), chamado Ginga. Deste então, foram desenvolvidas algumas aplicações utilizando esta linguagem. Assim, o grupo já acumulava certo conhecimento na utilização de Lua, mas não tinha, como foco, aplicações para *Web*. Portanto, visualizou-se que o desenvolvimeto do projeto usando HTML-CGILua-Lua, possibilitaria um aprofundamento do conhecimento sobre os diversos mecanismos e interfaces presentes nessa combinação.

É bom ressaltar que o trabalho de um grupo PET envolve a união das áreas de pesquisa, ensino e extensão. Assim, é interessante que sejam investigadas novas alternativas, diferentes das já dominadas, para tentar solucionar problemas ligados à Engenharia de Telecomunicações.

3 Desenvolvimento

O desenvolvimento do projeto compreendeu duas fases: i) a instalação do módulo CGILua no servidor Apache e ii) a criação das páginas dinâmicas usando código Lua embarcado em código HTML-CSS.

3.1 Instalação do módulo CGILua no servidor Apache

Inicialmente, foi preciso instalar o módulo CGILua no servidor *Web* utilizado (Apache 2.2), para que este servidor pudesse interpretar o código Lua embarcado nas páginas a serem desenvolvidas. Escolhemos o Apache porque este é o servidor que atende ao *website* do grupo, além do fato dele ser um dos mais utilizados em ambiente *Web*, o que facilita o aprendizado sobre o mesmo.

Em seguida, foi necessário obter e instalar os pacotes *lua-5.1.4.tar.gz* (interpretador Lua) e *luarocks-2.0.tar.gz* (ferramenta para instalação e gerenciamento de módulos Lua).

Após esta etapa, usamos o *luarocks* para instalar os módulos CGILua e WSAPI-FCGI.

Feito isso, foi necessário configurar o Apache para que ele pudesse carregar os módulos instalados e para que arquivos dos tipos "nome.lp" e "nome.lua" pudessem ser devidamente tratados. Para isso, precisamos editar o arquivo *httpd.conf* (Slackware) ou o arquivo *000-default* (Ubuntu), adicionando as diretivas *Options ExecCGI* e *AddHandler cgi-script .lua .lp* no campo respectivo ao nosso diretório.

Finalmente, bastou reiniciar o Apache para que as novas configurações fossem carregadas.

Todo esse procedimento é descrito, com detalhes, em documento disponível para *download* gratuito no *website* do grupo, na Seção *Downloads*.

3.2 Criação dinâmica das páginas

Os nomes dos arquivos contendo código Lua embarcado em código HTML possuem sintaxe *nome.lp*. Estes arquivos contêm código HTML comum e, em determinados pontos, é feita uma chamada ao código Lua. Isso se dá através da diretiva de abertura "*<?lua*", de forma que, a partir de então, o código Lua possa ser inserido. Para finalizar a seção reservada à Lua, basta inserir a diretiva de fechamento "*?>*".

Para facilitar o seu desenvolvimento e a sua manutenção, o programa Lua foi divido em módulos. Há três tipos de arquivos diferentes, que interagem entre si para a criação dinâmica das páginas: i) os arquivos *nome.lp*, que armazenam o código HTML-CGILua, ii) o arquivo *function.lua*, que armazena as funções Lua que são usadas para processar os dados do banco implementado e iii) os arquivos que implementam a base de dados, que são acessados pelas funções a fim de gerar o código *Web* final.

Todos os arquivos citados acima são do tipo texto.

Os arquivos da base de dados contêm tabelas Lua, que armazenam as informações relativas a cada integrante. Cada uma delas está inserida dentro de uma estrutura definida como *entry*{*tabela*}, como ilustrado no seguinte exemplo:

```
entry{
  Nome             = "Nome do Bolsista/Colaborador/Tutor",
  DatadeNascimento = {01,01,1990},
  Ingressopet      = {01,2000},
  Egressopet       = {01,2002},
  Ingressoies      = {01,2000},
}
```

Esta estrutura possui um significado particular em Lua. Primeiro, em Lua, as tabelas são criadas por meio do construtor "{ *conteudo_da_tabela* }". Além diso, normalmente, uma chamada de função é realizada com a sintaxe "*funcao(param1, param2, ..., paramN)*". Porém, em Lua, quando existe apenas um parâmetro na chamada da função, os parênteses podem ser omitidos. Assim, se for definida a função *entry(param)*, a sintaxe *entry*{*tabela*} significa a chamada da função *entry*, passando {*tabela*} como parâmetro.

Desse modo, a estrutura tem dois usos: registro de informação armazenado em arquivo e código Lua para chamada de uma função de processamento de dados.

Uma vez que, em Lua, as funções podem ser redefinidas dinamicamente, isso é um poderoso mecanismo de processamento de informação.

É importante observar que as informações relativas a datas são armazenadas como tabelas, dentro da tabela principal. Isto é possível devido à grande flexibilidade que a linguagem Lua oferece.

Com base nessa estrutura de dados, foram desenvolvidos códigos para a geração dinâmica de três páginas, que estão na Seção *Integrantes* do *website do grupo*, quais sejam: *informações.lp*, *aniversarios.lp* e *cronologia.lp*. Todas elas são decritas a seguir.

3.2.1 *Informações.lp*

A página a ser gerada pelo código presente neste arquivo deve apresentar informações sobre os tutores, os bolsistas e os colaboradores, que estão presentes ou que passaram pelo Grupo PET-Tele. As informações estão divididas por seções, tais como: tutor, bolsistas, voluntários, colaboradores, ex-tutores, ex-bolsistas, ex-voluntários e ex-colaboradores. Cada uma delas apresenta diversos dados, tais como: nome, ingresso e egresso do grupo, ingresso e egresso da IES, período atual do curso, emprego/estágio atual, departamento de origem e maior titulação acadêmica.

Os dados foram organizados em arquivos, de forma que cada um destes é associado a uma das citadas seções. Por exemplo, foi criado um arquivo *dados_bolsistas.lua*, que armazena informações relativas a todos os atuais bolsistas, assim como também foi criado um arquivo *dados_exbolsistas.lua*, com as informações dos ex-bolsistas, e assim por diante. Cada um desses arquivos é carregado pelo comando *dofile('nome_do_arquivo')* na sua respectiva seção, dentro de *informacoes.lp*.

Porém, antes de carregar o arquivo de dados a ser processado, deve-se definir uma função *entry()* específica, a qual está armazenada no arquivo *functions.lua*. Para tal, este último arquivo deve ser carregado antes da base de dados. Assim, quando a função *dofile()* é chamada para incluir a base de dados, o interpretador Lua entende que cada uma das tabelas presentes neste arquivo está sendo passada como argumento para cada uma das chamadas da função *entry()* definida. Isso é ilustrado no seguinte exemplo:

```
<?lua
 dofile('functions.lua')
 entry = listar_bolsistas
 dofile('dados_bolsitas.lua')
 ...
?>
```

A função *listar_bolsistas()*, usada no exemplo acima, é responsável por armazenar todos os nomes, endereços de *e-mail* e datas de ingresso no Grupo PET-Tele, em variáveis temporárias, que, então, podem ser inseridas na página *Web*. Isto é feito através do comando *print()*, que possui função similar ao comando *echo* do PHP, o que possibilita que números e *strings* que estão no ambiente Lua passem a fazer parte da página em HTML.

Tal mecanismo é mostrado no seguinte exemplo:

```
<?lua
 for i=1,#nomes do
   print('<tr><td>',nomes[i],'</a></td>')
 end
?>
```

Deve-se observar que *#nomes* é um comando que retorna o tamanho da tabela *nomes*, que, por sua vez, foi gerada pela função *listar_bolsistas*.

Este mecanismo foi usado em todas as seções da página a ser gerada por *integrantes.lp*, de forma a obter a maior parte das informações que são apresentadas pela mesma.

3.2.2 *Aniversarios.lp*

A página a ser gerada pelo código presente neste arquivo deve apresentar informações sobre as datas de aniversário dos tutores, dos bolsistas, dos voluntários, dos ex-bolsistas e dos ex-voluntários do Grupo PET-Tele, agrupadas por meses, que são organizados em uma tabela de quatro linhas por três colunas.

A tabela de datas foi criada usando *tags* HTML. Em cada espaço reservado para a inserção dos nomes e das datas, foi inserido código Lua para incluir os dados. Assim, tais informações são inseridas dinamicamente, durante o acesso ao *website*.

Como foi mostrado na seção anterior, o arquivo de funções *functions.lua* deve ser carregado e a função *entry()* deve ser definida, que, neste caso, foi a *mes_aniversario()*. Após este procedimento, os arquivos referentes aos dados dos tutores, dos bolsistas, dos voluntários, dos ex-bolsistas e dos ex-voluntários são carregados. Isso é mostrado no exemplo abaixo:

```
<?lua
 dofile('functions.lua')
 entry = mes_aniversario
 dofile('dados_tutores.lua')
 dofile('dados_bolsistas.lua')
 dofile('dados_voluntarios.lua')
 dofile('dados_exbolsitas.lua')
 dofile('dados_exvoluntarios.lua')
 ...
?>
```

A função *mes_aniversario()* tem o papel de testar cada entrada presente na base de dados (as tabelas dos integrantes) e armazenar o nome e o dia de aniversário de cada integrante em uma tabela associada ao correspondente mês. Obviamente, são 12 tabelas, nomeadas de acordo com os respectivos meses.

Após gerar as tabelas associadas aos respectivos meses, as informações armazenadas foram inseridas nas suas respectivas áreas, dentro do código da página, em ordem alfabética. Isto foi feito através do uso das funções *table.sort()* e *print()*, com um controle de fluxo adequado. Tal procedimento é exemplificado, para o mês Janeiro, no seguinte exemplo:

```
<?lua
 table.sort(Janeiro)
 for i,j in pairs(Janeiro) do
   print(j,'<br />')
 end
?>
```

O código acima foi utilizado ao longo de toda a página, para todos os meses seguintes.

3.2.3 *Cronologia.lp*

A página a ser gerada pelo código presente neste arquivo deve apresentar informações sobre os alunos que integraram o Grupo PET-Tele, em ordem cronológica, desde a criação do grupo até os dias atuais. Os nomes dos alunos são organizados em grupos de *mês/ano* nos quais houve alguma mudança na equipe, como entrada e/ou saída de bolsistas. Cada nome é apresentado dentro de um balão, com uma determinada cor de fundo. As cores verde, vermelho e branco, identificam os alunos ingresso, egresso e presente, respectivamente.

Para criar esta página dinamicamente, o programa foi separado em duas partes. A primeira tem o papel de analisar todas as entradas presentes nos arquivos referentes aos bolsistas e aos ex-bolsistas com o intuito de gerar uma lista (tabela) contendo todos os meses nos quais houve modificação (entrada e/ou saída) no quadro de alunos. A fim de organizar os respectivos meses em ordem cronológica, cada mês foi transformado em um número, que corresponde à diferença entre o número de segundos do primeiro dia deste mês e o primeiro segundo de 01 de Janeiro de 1970, de acordo com o fuso-horário da cidade de Greenwich, Inglaterra. Isto foi realizado com auxílio da função *os.time()*. Após identificados todos os meses nos quais ocorreram mudanças, foi criada uma tabela chamada *lista_meses*. Para cada um dos seus registros, a chave de acesso é um mês e o conteúdo é uma nova tabela, inicialmente vazia. Cada uma destas tabelas internas foi usada para agrupar os integrantes que estavam presentes no grupo no respectivo mês.

A segunda parte do código tem como objetivo mapear, para cada um dos meses listados, quais eram os integrantes que faziam parte do grupo, quais estavam entrando e quais estavam saindo. Primeiramente, a data de entrada do bolsista ou do ex-bolsista é convertida para número de segundos e armazenada na variável local *num_seg_i*. Logo após, é testada a data de saída. Caso seja encontrado o padrão *mês=00/ano=00*, isso significa que o aluno é bolsista atual e está presente no grupo. Portanto, a variável local *num_seg_e* recebe o valor 0. Caso contrário, esta variável armazena o número de segundos referente à data de saída. Em seguida, para cada mês, essas duas variáveis são comparadas com o número de segundos do mês em questão, e, dependendo do resultado, é preenchida a tabela interna relacionada a tal mês, com o nome e a cor do balão do integrante. Após este passo, a tabela *lista_meses* contém os índices associados aos meses que irão compor a página e as tabelas internas, que, por sua vez, contêm os nomes dos bolsistas e dos ex-bolsistas que estavam presentes, entrando ou saindo do grupo no mês correspondente.

De posse da informação organizada adequadamente, é preciso gerar a página. Para isso, todas as entradas desta tabela (já em ordem cronológica, do mais recente ao mais antigo) são acessadas com um controle de fluxo adequado (*for*), gerando uma seção dentro do documento HTML, que terá o nome do mês em questão. Para obter novamente os nomes dos meses e dos anos, foi usada a função *os.date()*, que faz o processamento inverso da função *os.time()*.

Por padrão, cada uma das seções é organizada em linhas que contêm quatro integrantes. Porém, pode haver um número variável de elementos em cada seção. Dessa forma, foi necessário realizar um controle adicional para gerar a formatação adequada dos dados.

4 Resultados

Os resultados obtidos foram considerados satisfatórios. Todas as informações que estavam presentes nas páginas que utilizavam a combinação de códigos HTML-CSS puderam ser reproduzidas usando a combinação HTML-CSS-CGILua-Lua. Porém, a implementação atual oferece algumas vantagens: i) a redundância de dados foi suprimida, pela criação de uma base de dados única, ii) os arquivos da base de dados são facilmente editáveis e portáveis, uma vez que são do tipo texto e iii) as páginas, que compartilham dados, são montadas automaticamente.

5 Conclusão e trabalhos futuros

O Grupo PET-Tele visualizou uma oportunidade de desenvolvimento, baseado em uma necessidade interna, mas que pode servir de solução para necessidades profissionais externas. Foi possível observar que o mecanismo proposto no início deste texto é realizável. Porém, todo o processo precisa ser lapidado em alguns aspectos. Como exemplo, a forma de acesso e de manipulação da base de dados é uma fonte potencial de erros, os quais podem danificar a página gerada. O programador das páginas deve ter consciência de que, ao inserir informações nos arquivos, deverá seguir os padrões da linguagem Lua, para que estes dados possam ser manipulados corretamente pelos programas. Também deve-se pensar em como criar mecanismos de proteção, como níveis de privilégio, visando proteger as informações presentes na base de dados.

Acreditamos que, para uma base de dados relativamente simples, que não exija um nível de proteção sofisticado, o mecanismo desenvolvido neste trabalho é uma boa opção, devido à sua simplicidade e ao poder computacional que a linguagem Lua oferece.

Como próximo passo no desenvolvimento desta ideia, pode-se propor a criação de uma interface que ofereça ferramentas para a criação e a edição de uma base de dados usando Lua e os mecanismos descritos neste trabalho. Esta interface deverá ter todas as funcionalidades que possuem os sistemas de gerenciamento de banco de dados existentes.

6 Anexos

Abaixo, é apresentado o conteúdo do arquivo *functions.lua*, que armazena todas as funções criadas ao longo do projeto.

```
--------------------------------

-- functions.lua: arquivo de funções
-- Criado em 18 de Agosto de 2012
-- Autor: Bruno Martins Costa - Engenharia de Telecomunicações - UFF

--------------------------------

nm      = {} -- armazena o nome dos bolsistas
em      = {} -- email dos bolsistas
per     = {} -- período dos bolsistas
atm     = {} -- onde está atualmente
ing_mes = {} -- mês de ingresso
ing_ano = {} -- ano de ingresso
egs_mes = {} -- mês de egresso
egs_ano = {} -- ano de egresso
orig    = {} -- origem colaboradores
tit     = {} -- título colaboradores

--------------------------------

-- meses de aniversário

Janeiro = {}
Fevereiro = {}
Marco = {}
Abril = {}
Maio = {}
Junho = {}
Julho = {}
Agosto = {}
Setembro = {}
Outubro = {}
Novembro = {}
Dezembro = {}

--------------------------------

meses = {'Janeiro', 'Fevereiro', 'Março', 'Abril', 'Maio', 'Junho', 'Julho', 'Agosto', 'Setembro', 'Outubro', 'Novembro', 'Dezembro'}

--------------------------------

function listar_bolsistas(arg)

  arg = conv_ascii_html(arg)

  nm[#nm+1]=arg.Nome
  ing_mes[nm[#nm]]=arg.Ingresso_pet[1]
  ing_ano[nm[#nm]]=arg.Ingresso_pet[2]
  per[nm[#nm]]=arg.Periodo
  em[nm[#nm]]=arg.Email
end


function listar_exbolsistas(arg)

  arg = conv_ascii_html(arg)

  nm[#nm+1]=arg.Nome
  ing_mes[nm[#nm]]=arg.Ingresso_pet[1]
  ing_ano[nm[#nm]]=arg.Ingresso_pet[2]
  egs_mes[nm[#nm]]=arg.Egresso_pet[1]
  egs_ano[nm[#nm]]=arg.Egresso_pet[2]
  em[nm[#nm]]=arg.Email
  atm[nm[#nm]] = arg.Atualmente
end


function listar_colaboradores(arg)

  arg = conv_ascii_html(arg)

  nm[#nm+1]=arg.Nome
  em[nm[#nm]]=arg.Email
  orig[nm[#nm]] = arg.Origem
  tit[nm[#nm]] = arg.Titulo
end


function listar_extutores(arg)

  arg = conv_ascii_html(arg)

  nm[#nm+1]=arg.Nome
  em[nm[#nm]]=arg.Email
  ing_mes[nm[#nm]]=arg.Inicio_tutoria[1]
  ing_ano[nm[#nm]]=arg.Inicio_tutoria[2]
  egs_mes[nm[#nm]]=arg.Termino_tutoria[1]
  egs_ano[nm[#nm]]=arg.Termino_tutoria[2]
end
```

```
function calc_per(arg)
--
  local mes_inicial = {1,7}
--

  arg = conv_ascii_html(arg)

  t1 = os.time()
  t2 = os.time{year=arg.Ingresso_ies[2], month=mes_inicial[arg.Ingresso_ies[1]], day=1}
  tdif = t1 - t2
  local temp = tdif/2628000 - (tdif%2628000)/2628000
  temp = (temp/6 - (temp%6)/6)+1
  if temp>10
    then
      if arg.Sexo == "f"
        then per[arg.Nome] = "Formada"
        else per[arg.Nome] = "Formado"
      end
    else per[arg.Nome] = tostring(temp)..'&ordm; Per&iacute;odo'
  end
end


function conv_ascii_html(arg2)

  dofile("tabela_asc_html.lua")

  for i,j in pairs(arg2) do
    if  type(j) == "string"
      then
        arg2[i] = string.gsub(arg2[i], '&', "&amp;")
        for r,s in pairs (tabela) do
          arg2[i] = string.gsub(arg2[i], r , s)
        end
      elseif type(j) == "table"
        then
          for u,v in pairs(j) do
            if type(v) == "string"
              then
                arg2[i][u] = string.gsub(arg2[i][u], '&', "&amp;")
                for r,s in pairs (tabela) do
                  arg2[i][u] = string.gsub(arg2[i][u], r , s)
                end
            end
          end
    end
  end
  return(arg2)
end


function mes_aniversario(arg)

arg = conv_ascii_html(arg)

if      arg.Data_de_Nascimento[2] == 01 then Janeiro[#Janeiro+1]     = '('..string.sub("0"..tostring(arg.Data_de_Nascimento[1]),-2)..') '..arg.Nome
elseif  arg.Data_de_Nascimento[2] == 02 then Fevereiro[#Fevereiro+1] = '('..string.sub("0"..tostring(arg.Data_de_Nascimento[1]),-2)..') '..arg.Nome
elseif  arg.Data_de_Nascimento[2] == 03 then Marco[#Marco+1]         = '('..string.sub("0"..tostring(arg.Data_de_Nascimento[1]),-2)..') '..arg.Nome
elseif  arg.Data_de_Nascimento[2] == 04 then Abril[#Abril+1]         = '('..string.sub("0"..tostring(arg.Data_de_Nascimento[1]),-2)..') '..arg.Nome
elseif  arg.Data_de_Nascimento[2] == 05 then Maio[#Maio+1]           = '('..string.sub("0"..tostring(arg.Data_de_Nascimento[1]),-2)..') '..arg.Nome
elseif  arg.Data_de_Nascimento[2] == 06 then Junho[#Junho+1]         = '('..string.sub("0"..tostring(arg.Data_de_Nascimento[1]),-2)..') '..arg.Nome
elseif  arg.Data_de_Nascimento[2] == 07 then Julho[#Julho+1]         = '('..string.sub("0"..tostring(arg.Data_de_Nascimento[1]),-2)..') '..arg.Nome
elseif  arg.Data_de_Nascimento[2] == 08 then Agosto[#Agosto+1]       = '('..string.sub("0"..tostring(arg.Data_de_Nascimento[1]),-2)..') '..arg.Nome
elseif  arg.Data_de_Nascimento[2] == 09 then Setembro[#Setembro+1]   = '('..string.sub("0"..tostring(arg.Data_de_Nascimento[1]),-2)..') '..arg.Nome
elseif  arg.Data_de_Nascimento[2] == 10 then Outubro[#Outubro+1]     = '('..string.sub("0"..tostring(arg.Data_de_Nascimento[1]),-2)..') '..arg.Nome
elseif  arg.Data_de_Nascimento[2] == 11 then Novembro[#Novembro+1]   = '('..string.sub("0"..tostring(arg.Data_de_Nascimento[1]),-2)..') '..arg.Nome
elseif  arg.Data_de_Nascimento[2] == 12 then Dezembro[#Dezembro+1]   = '('..string.sub("0"..tostring(arg.Data_de_Nascimento[1]),-2)..') '..arg.Nome
end

end

-------------------------------

num_seg_mes = {}
lista_meses = {}


function cron_pet (arg)

  local num_seg_i = os.time{year=arg.Ingresso_pet[2], month=arg.Ingresso_pet[1], day=1}
  local num_seg_e

  if arg.Egresso_pet[1] == 0 or arg.Egresso_pet[2] == 0
    then num_seg_e = 0
    else num_seg_e = os.time{year=arg.Egresso_pet[2], month=arg.Egresso_pet[1], day=1}
  end
  if #num_seg_mes == 0
    then
      if num_seg_e ~= 0
        then
          num_seg_mes = {num_seg_i , num_seg_e}
        else
          num_seg_mes = {num_seg_i}
      end
    else
      for i,j in pairs(num_seg_mes) do
        if j == num_seg_i
          then break
          elseif i == #num_seg_mes
            then
```

```
            num_seg = os.time{year=arg.Ingresso_pet[2], month=arg.Ingresso_pet[1], day=1}
            num_seg_mes[#num_seg_mes+1] = num_seg_i
        end
      end
      if num_seg_e ~= 0
        then
          for i,j in pairs(num_seg_mes) do
            if j == num_seg_e
              then break
              elseif i == #num_seg_mes
                then
                  num_seg = os.time{year=arg.Ingresso_pet[2], month=arg.Ingresso_pet[1], day=1}
                  num_seg_mes[#num_seg_mes+1] = num_seg_e
            end
          end
      end
  end
end


function cron_pet2 (arg)

-- teste dos bolsistas e exbolsistas para obter a sua estadia no pet

  local num_seg_i = os.time{year=arg.Ingresso_pet[2], month=arg.Ingresso_pet[1], day=1}
  local num_seg_e

  if arg.Egresso_pet[1] == 0 or arg.Egresso_pet[2] == 0
    then num_seg_e = 0
    else num_seg_e = os.time{year=arg.Egresso_pet[2], month=arg.Egresso_pet[1], day=1}
  end
  for i,j in pairs(lista_meses) do
    if num_seg_i == i
      then
        j[#j+1] = '<div class="balao Verde"><div class="nome">'..arg.Nome..'</div></div>'
      elseif num_seg_i < i and (num_seg_e > i or num_seg_e == 0)
        then
        j[#j+1] = '<div class="balao Branco"><div class="nome">'..arg.Nome..'</div></div>'
        elseif num_seg_e == i
          then
            j[#j+1] = '<div class="balao Vermelho"><div class="nome">'..arg.Nome..'</div></div>'
    end
  end
end
```