Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

Programação em Sistemas Computacionais

Semestre de Verão de 2015/2016

Série de Exercícios 3 - Trabalho de Grupo

Construa os programas e bibliotecas indicados na Parte II, usando a linguagem C, tendo o cuidado de eliminar repetições no código fonte e de isolar funcionalidades distintas em diferentes ficheiros fonte. Entregue o código desenvolvido, devidamente indentado e comentado, bem como o `Makefile` para gerar os executáveis e bibliotecas a partir do código fonte. Assegure-se de que o compilador não emite qualquer aviso sobre o seu código, mesmo com a opção **`-Wall`** activa, e de que no final da execução do programa não existem recursos por libertar (memória alocada dinamicamente e ficheiros abertos).

Apresente um relatório com a descrição sucinta das soluções, que deverá ser um guia para a compreensão do código desenvolvido e não a tradução do código para língua natural. Contacte o docente sempre que tiver dúvidas.

Encoraja-se a discussão dos problemas e das respectivas soluções com colegas de outros grupos (tenha em consideração que a partilha directa de soluções implica, no mínimo, a anulação das entregas dos grupos envolvidos).

O produto final desta série de exercícios é uma biblioteca que implemente serviços de consulta de informação meteorológica e um programa que utilize a funcionalidade implementada pela biblioteca. A informação meteorológica pode ser obtida no serviço disponibilizado pela *Dark Sky Company*, utilizando uma API *Web*. Toda a informação sobre este serviço pode ser obtida a partir do *site* forecast.io.

Embora o enunciado esteja organizado em 6 pontos, é feita apenas uma entrega final, com o código fonte da biblioteca (ponto 5), o programa de utilização (ponto 6) e um *Makefile* que defina os *targets* **`lib`**, **`app`** e **`clean`**, que têm como finalidade, respectivamente, geração da biblioteca, geração do programa para utilização e elimiação dos artefactos gerados pelos *targets* anteriores.

Parte I - Preparação do ambiente de desenvolvimento

O acesso à informação usa o protocolo HTTP (*Hypertext Transfer Protocol*) para aceder aos serviços definidos por uma API REST (*Representational State Transfer*). É prática comum que os serviços suportados em APIs REST usem o paradigma pergunta/resposta. A pergunta é representada pelo URI (*Uniform Resource Identifier*) que é usado no pedido HTTP GET enviado ao servidor; a resposta ao pedido é codificada no formato JSON (*JavaScript Object Notation*), de acordo com o esquema definido pela API.

A documentação da API está disponível em https://developer.forecast.io/docs/v2. A *Dark Sky Company* controla o acesso ao serviço através de uma chave de cliente (`APIKEY`) que deverá ser embutida no URI usado em cada pedido HTTP. Para obter a chave de cliente é necessário proceder ao registo no *site* (uma chave gratuita permite realizar 1000 pedidos por dia).

CURL

O acesso ao serviço faz-se estabelecendo uma ligação ao servidor usando o protocolo HTTP. Para

suportar as comunicações com o servidor deverá ser utilizada a biblioteca *open source* **`libcurl`**.

Instalação: **`$ sudo apt-get install libcurl4-gnutls-dev`**

Documentação: http://curl.haxx.se/libcurl

JSON

Para interpretar as respostas do servidor em formato JSON deverá ser utilizada a biblioteca *open source* **`jansson`**.

Instalação:

1. Descarregar o código fonte mais recente (pacote jansson-2.7.tar.gz).
2. Descompactar: **`$ tar -xzvf jansson-2.7.tar.gz`**
3. Tendo como diretório corrente **`jansson-2.7`**, executar:

   `$ ./configure ; make ; sudo make install`

Documentação: https://jansson.readthedocs.io/en/2.7/index.html

Tal como nas linguagens de programação, a indentação da escrita em formato JSON facilita a leitura direta por parte do humano. A *Google* disponibiliza, em https://json-indent.appspot.com/, um serviço que recebendo o URI de um recurso JSON apresenta o respectivo conteúdo indentado.

Valgrind

Para verificar se o programa liberta toda memória que alocou dinamicamente, deve utilizar-se a ferramenta **`valgrind`**.

Instalação: **`$ sudo apt-get install valgrind`**

Documentação: **`$ man valgrind`**

Parte II - Realização

1. Recorrendo à biblioteca **libcurl**, implemente a função **http_get** que realiza um pedido HTTP GET ao URI especificado através do parâmetro **uri** e armazena o resultado no ficheiro cujo nome é especificado através do parâmetro **filename.** A função devolve um booleano (0 ou 1) que indica se houve sucesso.

   ```
   int http_get(const char *uri, const char *filename);
   ```

   Escreva um programa de teste que, recebendo como argumentos um URI e o nome de um ficheiro, permita verificar o correto funcionamento desta função, descarregando o conteúdo do recurso para o ficheiro. Pode experimentar com o URI: http://imagem.band.com.br/f_198156.jpg.

2. Utilizando as bibliotecas **libcurl** e **jansson**, implemente a função **http_get_json_data**, que realiza um pedido HTTP GET ao URI especificado através do parâmetro **uri**, que deve corresponder a um recurso HTTP do tipo **application/json** e retorna o ponteiro para uma instância do tipo **json_t** (definido pela biblioteca **jansson**) com o conteúdo da resposta. Se ocorrer um erro durante a transferência, a função retorna **NULL** e imprime no **stderr** a mensagem que indica a razão do erro.

   Não deve ser usado um ficheiro temporário, isto é, a resposta ao pedido HTTP GET deve ser mantida em memória.

   ```
   json_t * http_get_json_data(const char *uri);
   ```

   O programa de teste pode, por exemplo, aceder ao recurso cujo URI é https://api.forecast.io/forecast/<APIKEY>/38.722252,-9.139337 e afixar na consola a informação meteorológica atual. **APIKEY** é a chave fornecida pela *Dark Sky*; os números finais são as coordenadas geográficas (latitude e longitude) de Lisboa.

3. Utilizando a função do ponto anterior, implemente a função **get_weather** para obter a informação meteorológica do local e data especificados como argumentos.

   O URI usado no pedido HTTP GET é construído segundo o formato *Time Machine* https://api.forecast.io/forecast/<APIKEY>/<LATITUDE>,<LONGITUDE>,<TIME>, onde: **APIKEY** pode ser embutida no código do programa; **TIME** é escrito no formato **[YYYY]-[MM]-[DD]T[HH]:[MM]:[SS]** e representa uma hora GMT. Para usar horas locais deve acrescentar-se um sufixo na forma **{+,-}[HH][MM]**, que corresponde ao desvio em relação à hora GMT. Em Portugal na hora de verão o desvio é `+0100`, sendo a hora de inverno igual à hora GMT.

   ```
   typedef struct weather {
       float min_temp;
       float max_temp;
       float wind;
       float humidity;
       float cloud;
   } Weather;

   typedef struct location {
       const char *name;
       float latitude;
       float longitude;
   } Location;

   typedef struct date {
   ```

```
    unsigned int year;
    unsigned short month;
    unsigned short day;
    unsigned short hour;
    unsigned short minute;
    unsigned short second;
    short deviation_hour;
    unsigned short deviation_minute;
} Date;

weather *get_weather(Location *location, Date *date);
```

Realize um programa de teste que recebendo como argumentos as coordenadas dum local e a informação data/hora mostre na consola o resumo diário da informação meteorológica (contida nos campos **`daily/data`** da resposta).

4. Utilizando a função do ponto anterior, implemente a função **`get_weathers`** que recebendo uma colecção de coordenadas geográficas e informação data/horária e devolva uma colecção com as informações meteorológicas relativas aos locais especificados.

   ```
   Weathers get_weathers(Locations locations, Date *date);

   void free_weathers(Weathers weathers);
   ```

   Defina os tipos de dados colecção **`Weathers`** e **`Locations`** de forma adequada. A função **`free_weathers`** deve libertar os recursos de memória alocados dinamicamente aquando da construção da colecção retornada pela função **`get_weathers`.** Escreva um programa de teste que apresente a informação recebida na consola em forma de tabela.

5. Construa uma biblioteca de ligação dinâmica (*shared object*) com as funções desenvolvidas nos pontos anteriores e com as funções auxiliares que entender necessárias. Na organização do código, tenha em consideração que deve evitar repetições de código fonte.

6. Desenvolva um programa que, utilizando a biblioteca produzida no ponto anterior, obtém a informação meteorológica sobre um conjunto de locais para depois apresentá-la na forma de uma tabela ordenada. As coordenadas dos locais são fornecidas como o conteúdo de um ficheiro em formato CSV (*Comma-Separated Values*) onde, relativamente a cada local, são definidos os campos **`<latitude>`**, **<longitude>**, **<local name>**. As coordenadas geográficas de um local podem ser obtidas no *Google Maps* ou no *site* http://www.gps-coordinates.net/.

   Os critérios de ordenação são definidos como argumentos do programa e consistem na especificação da grandeza física - temperatura mínima, temperatura máxima, vento, humidade, nebulosidade - e da ordem - crescente ou decrescente.

   Exemplos de invocação do programa e respectivo resultado.

   ```
   $ weather
   usage: weather <options> <locations_file> <date>
   options:
   -a : ascending order
   -d : descending order
   -f M : maximum temperature
   -f m : minimum temperature
   -f u : humidity
   -f n : cloud cover
   -f v : wind speed
   ```

```
$ weather -f n -c coord.csv 1974-4-25
Date: 1974-4-25
           Min. Temp     Max. Temp     Wind          Humidity      Clouds
Faro       7.9           17.2          7.33          0.69          0.56
Porto      7.2           16.1          1.69          0.87          0.66
Coimbra    7.2           16.1          0.87          0.85          0.78
Évora      7.6           17.2          7.06          0.75          0.81
Lisboa     10.0          14.9          5.07          0.75          0.91
```

Data limite de entrega: 12 de Junho de 2016

ISEL, 6 de Maio de 2015

Carlos Martins, Ezequiel Conde

Referências

HTTP http://www.w3.org/Protocols/rfc2616/rfc2616.html
JSON http://www.json.org/
CSV https://tools.ietf.org/html/rfc4180