

Construa os programas e a biblioteca 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 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). Deve verificar essa situação com o utilitário Valgrind. Em caso de erro irreversível, o programa não deve terminar descontroladamente, deve, no mínimo emitir, uma mensagem informativa e em seguida terminar. O código desenvolvido será valorizado segundo os seguintes critérios, em importância decrescente: correção, eficiência, clareza.

Encoraja-se a discussão dos problemas e das respectivas soluções com outros colegas (tenha em consideração que a partilha directa de soluções implica, no mínimo, a anulação das entregas dos alunos 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 companhia OpenWeather global services, utilizando uma API Web. Toda a informação sobre este serviço pode ser obtida a partir do site <https://openweathermap.org/api>.

Embora o enunciado esteja organizado em 6 pontos, a entrega pode apenas conter o código fonte da biblioteca (ponto 5), o programa de utilização (ponto 6) e um *Makefile* que define os *targets* **lib**, **app** e **clean**, que têm como finalidade, respectivamente, a geração da biblioteca, a geração do programa para utilização e a eliminação dos artefactos gerados pelos outros *targets*.

---

## 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 API REST usem o paradigma pergunta/resposta. A pergunta é representada pelo URL (*Uniform Resource Locator*) 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 a utilizar está disponível em [Solar system OpenData](#).

### CURL

O acesso ao serviço é feito estabelecendo ligações 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: [libcurl - the multiprotocol file transfer library](#).

### JSON

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

Instalação: **\$ sudo apt-get install libjansson-dev**

Documentação: [Jansson Documentation](#)

Tal como noutras linguagens, a indentação da escrita em formato JSON facilita a leitura directa por parte do humano. Sugere-se a utilização de um visualizador JSON, que pode ser instalado no *browser* na forma de *plug-in*.

# Valgrind

Para verificar se um programa liberta toda a memória alocada dinamicamente deverá utilizar a ferramenta **valgrind**.

Instalação: `$ sudo apt-get install valgrind`

Documentação: `$ man valgrind`

---

## Parte II - Realização

1. Utilizando a biblioteca **libcurl**, implemente a função **http\_get**, que realiza um pedido HTTP GET ao URL especificado através do parâmetro **url** e armazena o resultado num ficheiro cujo nome é especificado através do parâmetro **filename**. Se ocorrer algum erro durante a transferência, a função retorna um valor negativo e escreve em **stderr** a mensagem que indica a razão do erro.

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

Escreva um programa de teste que, recebendo como argumentos um URL e o nome de um ficheiro, permita verificar o correto funcionamento desta função, descarregando o conteúdo do recurso para o ficheiro. Teste o programa usando o URL <https://img.ibxk.com.br/2017/01/26/26115632211129.jpg?w=1040>.

2. Utilizando as bibliotecas **libcurl** e **jansson**, implemente a função **http\_get\_json\_data**, que realiza um pedido HTTP GET ao URL especificado através do parâmetro **url**, 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 mostra 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 desde a sua receção até não ser mais necessária.

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

O programa de teste pode, por exemplo, aceder ao recurso cujo URL é <https://api.openweathermap.org/data/2.5/onecall?lat=38.72&lon=-9.14&exclude=minutely.hourly.daily.alerts&appid={API Key}> e afixar na consola a temperatura atual na cidade de Lisboa. {API Key} é a chave fornecida pela OpenWeather.

3. Utilizando a função do ponto anterior, implemente a função **weather\_get** para obter a informação meteorológica de um local numa dada hora.

```
Weather *weather_get(Date *date, Location locations);
```

Utilize os seguintes tipos de dados:

```
typedef struct date {  
    int day;  
    int month;  
    int year;  
} Date;
```

```
typedef struct weather {  
    float temperature;  
    float wind_speed;  
    float humidity;  
    float cloud_cover;  
} Weather;
```

```
typedef struct location {  
    float latitude;
```

```

    float longitude;
    char *name;
} Location;

```

Utilize um programa de teste que afixe na consola a previsão meteorológica para a cidade de Lisboa para o dia seguinte.

- Utilizando a função do ponto anterior, implemente a função **weather\_get\_info** que, recebendo as coordenadas geográficas de vários locais e uma data, devolve a informação meteorológica desses locais nessa data. **<location info>** é um tipo de dados, a definir pelo estudante, que permita representar a informação geográfica de um conjunto variável de locais. **<weather info>** é um tipo de dados, a definir pelo estudante, que permita representar a informação meteorológica associada a um conjunto de locais. A informação contida neste tipo de dados deve ser auto-suficiente, isto é, não ser necessária informação externa para interpretar estes dados.

```

<weather info> weather_get_info(Date *date, <location info> locations);

```

Implemente também a necessária função **weather\_free\_info** que deve libertar os recursos de memória alocados dinamicamente aquando da construção da instância **<weather info>** retornada pela função **weather\_get\_info**.

Escreva um programa de teste que apresente a informação recebida na consola. Use o Browser para obter a mesma informação e compare os resultados.

- Construa uma biblioteca de ligação dinâmica (*shared object*) com as funções definidas 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.
- Desenvolva um programa que, utilizando a biblioteca produzida no ponto anterior, crie um ficheiro em formato CSV, com a informação meteorológica de um conjunto de locais, numa dada data.

O programa deve receber na linha de comando, a data e o nome de um ficheiro de texto com a lista dos locais.

```

$ weather <date> <location>

```

Data limite de entrega: 24 de Janeiro de 2020

ISEL, 8 de Dezembro de 2020