CES12: Instalação e dicas sobre as ferramentas

Instalação

Packages a serem instalados: (testado em Ubuntu 18.04 e 18.10)

```
sudo apt-get install build-essentials cmake git
```

build-essentials inclui o g++ e libs necessarias para C++
Cmake é a ferramenta de compilação
Git é necessário para automaticamente baixar o gtest, a ferramenta de teste

CMake

Conceito:
O arquivo CMakeLists.txt contém instruções para gerar Makefiles (ferramenta Make) para compilação. Vantagens:

1. Muito fácil compilar “off the source”. I.e., todos os arquivos gerados ficam em outro diretório, sem nunca mexer nos diretórios de fontes (tanto fornecidas pelo professor quanto escritas pelo aluno). Assim, para garantidamente “resetar” a compilação, basta deletar o diretório onde está o build.
2. Todas as instruções de compilação estão em arquivo textos. Nada está escondido em abas e opções perdidas em janelas e menus de configuração às vezes redundantes e conflitantes.
3. Toda a compilação está no console, e pode usar a IDE que desejar para editar.

Gtest

O Gtest é uma ferramenta que fornece uma série de macros e funções para que sejam definidos testes, que chamarão o seu código e compararão resultados com os resultados corretos.

O Gtest **não é uma lib**, *é um código que deve ser compilado junto com o seu código* - é dessa forma para que o gtest e o código da implementação sejam compilados com as mesmas opções de compilação, o que evita inconsistências nos testes.

Seria possível instalar fontes do gtest via packages da distribuição ubuntu, mas é mais fácil instruir o cmake a baixar a sua própria versão local de gtest - e é a solução recomendada.

Compilação comentada:

Downloaded lab1hash.zip, descomprimi.

```
lgm@lgmita:~$ cd lab1hash
lgm@lgmita:~/lab1hash$ ls
CMakeLists.txt   CMakeLists.txt.in    data    test   src
```

CMakeLists.txt são as instruções de compilação. CMakeLists.txt.in são instruções para baixar o gtest (o cmake se encarrega disto)
A seguinte estrutura de diretórios está hardcoded e deve ser mantida:

1. `data`: em alguns labs são fornecidos dados de entrada ou saída
2. `test`: código de testes, fornecido pelo professor
3. `src`: diretório onde o aluno deve escrever o seu código. Normalmente contém código com interfaces e cabeçalhos vazios que o aluno deve completar
    a. ATENÇÃO: o cmake está programado para preparar a compilação de qualquer arquivo **com extensão .cpp** neste diretório.
    b. Deve submeter um zip **deste** diretório.

```
lgm@lgmita:~/lab1hash$ mkdir build
```

Crio diretório para compilar. **A compilação nunca modifica nada em src, lib, ou test**.

```
lgm@lgmita:~/lab1hash$ cd build
lgm@lgmita:~/lab1hash/build$
```

E entro no diretório build recém criado.

```
lgm@lgmita:~/lab1hash/build$ cmake ..
```

Note que "`..`", tanto em unix quanto em windows, significa "diretório pai". Significa que o CMake procurará um arquivo chamado CMakeLists.txt no diretório pai do diretório corrente - que neste caso é exatamente o CMakeLists.txt fornecido. Mas toda a saída será gerada no diretório corrente, ou seja, em `build`.

Na primeira vez, há uma demora devido ao download do gtest. Haverá erros se o compilador C++, ou o git, não estiverem instalados.

```
lgm@lgmita:~/lab1hash/build$ ls
bin   make_install.cmake   googletest-src Makefile
 CMakeCache.txt   googletest-build    'lab1tests[1]_include.cmake'
```

```
 CMakeFiles      googletest-download   lib
```

Veja que além de baixar o gtest, foram criados vários arquivos. Agora é a compilação de verdade:

```
lgm@lgmita:~/lab1hash/build$ make
```

Na primeira vez é demorado pois o gtest é compilado. Uma lista de targets aparece sendo compilada até completar [100%].

```
lgm@lgmita:~/lab1hash/build$ ls
 bin  googletest-build      'lab1tests[1]_include.cmake'
 CMakeCache.txt       googletest-download  'lab1tests[1]_tests.cmake'
 CMakeFiles           googletest-src        lib
 cmake_install.cmake   lab1tests
```

Há um novo arquivo “lab1tests” que é o executável que inclui o seu código que (espera-se que) implementa hashing, e inclui também os testes fornecidos pelo professor, e o gtest. Assim, podemos chamar o executável, e ele chamará os testes sobre o seu código:

```
luiz@lgmita:~/lab1hash/build$ ./lab1tests
Running main() from
/home/lgm/lab1hash/build/googletest-src/googletest/src/gtest_main.cc
[==========] Running 5 tests from 2 test suites.
[----------] Global test environment set-up.
[----------] 4 tests from Lab1Hash
[ RUN      ] Lab1Hash.AddOneElement
[     OK ] Lab1Hash.AddOneElement (0 ms)
[ RUN      ] Lab1Hash.Add4Elements
[     OK ] Lab1Hash.Add4Elements (0 ms)
[ RUN      ] Lab1Hash.AddFileRandom
[     OK ] Lab1Hash.AddFileRandom (1 ms)
[ RUN      ] Lab1Hash.AddFilelength8
[     OK ] Lab1Hash.AddFilelength8 (0 ms)
[----------] 4 tests from Lab1Hash (2 ms total)

[----------] 1 test from AuxLab1Hash
[ RUN      ] AuxLab1Hash.HashValueIsInsideVector
[     OK ] AuxLab1Hash.HashValueIsInsideVector (0 ms)
[----------] 1 test from AuxLab1Hash (0 ms total)

[----------] Global test environment tear-down
[==========] 5 tests from 2 test suites ran. (2 ms total)
```

```
[  PASSED  ] 5 tests.
```

Note que o aluno não precisa escrever uma main(), pois o executável gerado pelo gtest já inclui uma main() fornecida pelo gtest - e o que esta main() faz é chamar os testes em cima do seu código e gerar esta saída dizendo que tudo passou (ou não!).

Mais ainda, como um executável não pode ter duas funções main(), *se o aluno incluir uma main(), o código não compila*. Quem desejar incluir uma main() para experimentar alguma coisa, deve compilar por outros meios e depois apagar ou comentar a main() antes de submeter.

Ah! Note o "`./`" em `./lab1tests`
"." significa "diretório corrente", e é necessário, a não ser que inclua "." no path da sua shell.

E, novamente, **A compilação nunca modifica nada em src, lib, ou test**.
Isso implica que, se quiser voltar para a estaca zero e compilar tudo novamente, basta deletar o diretório build com o seguinte comando (estando no diretório pai do build):

```
lgm@lgmita:~/lab1hash/$ rm -fr build
```

-f significa "force", para remover diretórios inteiros
-r significa "recursive" para remover todos os subdiretórios dentro de build.

Submissão:

Forneci um script bash chamado zipaParaEntregar

```
if [ -z "$1" ]
   then
    echo "ENTRE SEU NOME COMO ARGUMENTO, SEM ACENTOS, SEM ESPACOS.
EXEMPLO:"
    echo '$ ./zipaParaEntregar ZeJoaoSilva'
    exit
fi

zip -r ${1}.zip src/
```

O If apenas checa se forneceu uma string como argumento, que será o nome de um arquivo .zip contendo apenas o diretório src.
A vida de um programador bash é bem mais fácil se strings não tem espaços ou acentos…
Para usar o script é preciso estar no diretório raiz do lab (pai do diretório src)

```
lgm@lgmita:~/lab1hash/$ ./zipaParaEntregar ZeJoaoSilva
```

Naturalmente, a idéia deste script é apenas gerar o que é preciso para submeter:
Um zip chamado SeuNome.zip, contendo apenas o diretório src.

Zipe o diretório, não só os arquivos dentro do diretório. É impossível esquecer um arquivo se fizer assim, e o meu script de correção espera um diretório src dentro do seu zip. Em windows, equivale a selecionar o diretório e mandar zipar com o botão direito do mouse, e *não* a selecionar todos os arquivos, e usar o botão direito para zipar todos eles.

O relatório deve ser um pdf incluído dentro do mesmo zip (basta copiar o pdf para o diretorio src antes de zipar). Se quiser incluir varios arquivos relacionados com o relatório, crie um diretorio "src/relatorio" dentro do /src. De novo, todos os arquivos com extensão diferente de .cpp em /src não atrapalham a compilação.

SUBMETA APENAS UM (1) ARQUIVO ZIP, POIS O GOOGLECLASSROOM NÃO FAZ UM BOM TRABALHO EM SEPARAR AS SUBMISSÕES DOS ALUNOS.

STL não é magia negra, são estruturas de dados.

Veja este exemplo real de um lab em 2019 (não é o primeiro lab, não assuste se não entende OO suficiente ainda). A primeira tarefa do método era descobrir o valor máximo entre a,b,c. Com a seguinte implementação:

```
1 classe::MetodoNoLoopMaisInterno(int a, int b, int c) {
2     ctd::vector v;   //  objeto vetor alocado, mas nenhum espaco
                       //  alocado para elementos.
3     v.push_back(a);  // 1 elemento - aloca espaco para 1 elemento
4     v.push_back(b);  // 2 elementos - free(size 1), aloca(size 2)
5     v.push_back(c);  // 3 elementos - free(size 2), aloca(size 4)
6     std::sort(v);    // ordenação não aloca ou desaloca nada
7     int max = v[0];
8     // continua o metodo com outras tarefas
9 }
```

Veja que existem 4 mallocs e 2 frees nas linhas 2 a 5. Isto porque a política de alocação usual da classe std::vector quando um tamanho maior é requisitado é sempre dobrar o tamanho. De tamanho 0, passou para 1 (aqui não tem como dobrar), de tamanho 1, passou para 2, e enfim de 2 para 4.
Mas, internamente os dados estão em um vetor, ou seja, estão contíguos na memória. Portanto, dobrar o tamanho significa alocar um espaço 2x maior, copiar os dados, e depois liberar (free()) o espaço previamente alocado.

Como este método estava no loop mais interno do algoritmo, simplesmente o tempo de execução **triplicou**!
Veja como deve ser:

```
////////////// Declaracao da classe (.h)
class classe() {
     // all public stuff
     private:
     std::vector v; // atributo v criado vazio junto com a classe
     // other private stuff
}

////////////// implementação da classe, (.cpp)
classe::classe() { // construtor
     v.resize(3); // espaco para 3 elementos alocados no construtor
     // outras tarefas do construtor
 }

classe::MetodoNoLoopMaisInterno(int a, int b, int c) {
     // v é um atributo da classe, e espaco pre-alocado no construtor
     v[0] = a; v[1] = b; v[2] = c; // nenhuma alocacao, so atribuicao
     std::sort(v);
     int max = v[0];
     // continua o metodo com outras tarefas
 }
```

Óbvio que não precisa de um vetor para ordenar 3 elementos, e mesmo a solução ‘correta’ ainda é mais lenta do que usar uma série de ifs para encontrar o máximo. Mas o importante é que, como o método estava no loop mais interno do algoritmo, a diferença entre as 2 implementações é o triplo do tempo de execução.
Sempre lembre que dentro de std::vector há um vetor, não há magia negra e ***o que é fácil ou difícil de fazer com um vetor continua válido com std::vector***.
Outro exemplo, std::foward_list apenas encapsula uma lista ligada simples, quer dizer, a busca é linear e não O(logn).

Dicas Aleatórias sobre a STL

Ou, 2019 bloopers… não repita…

vector.resize(n) cria n elementos, não precisa inicializá-los

Exemplo:

```
1 std::vector<std::forward_list> v; // um vetor de listas ligadas
2 v.resize(10);// aloca espaco p/ 10 listas, chama constructor default
3 v[0].empty() // retorna true, sem erro: a lista 0 existe,está vazia.
```

O método resize aloca espaço para 10 objetos std::forward_list e chama o construtor default, sem argumentos. Ou seja, depois do resize, já existem 10 listas vazias.
Ou seja, na 3a linha v[0].empty() chama o metodo empty da primeira lista ligada.

PS: obviamente precisaria inicializar cada elemento se fosse um vetor de ponteiros.
E.g.:

```
1 std::vector<int*> vi;
2 std::vector<std::forward_list*> vfl;
3 vi.resize(10); vfl.resize(10);
```

No caso acima, os vetores apenas alocaram espaço para 10 ponteiros. Nenhum inteiro ou lista ligada foram criados ou alocados. vi[1] ou vfl[1] são ponteiros. Poderia agora fazer algo como:

```
int i = 10;
vi[0] = &i;
vfl[0] = new std::forward_list();
```

O que não deve ser estranho - basta pensar no que acontece quando alocamos um vetor de ponteiros na linguagem C.

Não precisa de nó cabeça, já existe um objeto lista que pode ser vazia.

Alguns aprenderam a implementar um nó cabeça para listas ligadas. O propósito do nó cabeça era facilitar e uniformizar o código em relação a representação de listas vazias, evitando verificações de ponteiro NULL, já que existe uma 'lista vazia'. Tudo bem, mas não há nenhuma necessidade de nó cabeça, ou marcador de vazio, ao usar a STL, já que, mesmo que a lista esteja vazia, existe o objeto lista.

```
std::forward_list<tipo> lista; // a lista é criada vazia
lista.empty();           // retorna true
lista.push_front(dado);  // funciona mesmo com a lista vazia
```

Não tem método size() na lista ligada?

Não. Mas há uma forma genérica válida para várias estruturas de dados, com uma sintaxe que inclusive deixa claro que custa O(n), usando std::distance(). Pesquise o que faz isso e como usar, procure exemplos. Ou, se não tiver curiosidade, basta percorrer a lista e contar os elementos.

FAQ

Cmake x make: chame cmake ao criar novos arquivos fonte!

Cmake gera os makefiles que o make utiliza para compilar, mas o Cmake precisa saber quais os arquivos a serem compilados.
Assim, se apenas mudar o conteúdo de um .cpp, não precisa chamar cmake de novo, basta chamar make e todos os arquivos modificados e suas dependências serão recompilados.

Mas, ao criar um novo arquivo `.cpp`, os makefiles não sabem que ele existe. É preciso chamar cmake novamente - o cmake procura arquivos .cpp no diretorio src.

E as minhas instruções do cmake apenas compilam arquivos com extensão `.cpp` no diretório src. (E **não** compilam outras extensões como `.c`, `.cp`, `.C`, `.cc`, etc.)

Quero usar a minha IDE favorita!

Crie um projeto em outro diretório fora da minha árvore de diretório. Nesse projeto, pode haver um main.cpp. Inclua nesse projeto o diretorio src, e edite os fontes no diretorio src. Outras tralhas que a IDE gerar no diretorio src serão ignoradas pela minha estrutura de compilação (a única coisa que o cmake procura são arquivos .cpp)
Também deve ser possível integrar o gtest na sua IDE (procure algum help em forums) e executar inclusive os meus testes direto na sua IDE.
Outra dica: a IDE cLion, que muitos na ITA Androids usam, é bem integrada com cmake, alunos de anos anteriores já conseguiram editar o código, chamar o cmake e executar os testes sempre a partir do CLion. (mas nunca usei porque o meu email ita não funcionou pro pedido de licença-professor!)

Mas antes de entregar, *certifique-se que a minha infraestrutura consegue compilar o seu diretório src*.

Como compilar vários .cpp? Header A inclui Header B que inclui Header A e esse #include recursivo não consigo compilar!

Problema conhecido, usual em C, e não apenas em C++:
http://faculty.cs.niu.edu/~mcmahon/CS241/c241man/node90.html
https://en.wikipedia.org/wiki/Include_guard
https://gcc.gnu.org/onlinedocs/cpp/Once-Only-Headers.html
https://en.wikibooks.org/wiki/More_C%2B%2B_Idioms/Include_Guard_Macro

Não tem main()? Pq a minha função main() não compila?

O gtest possui uma lib com função main() que é linkada junto com o seu código e o meu código. Se quiser criar uma função main para seu próprio uso durante o desenvolvimento, crie um projeto em alguma IDE e escreva a sua main() em outro diretorio.

Mas a sua função main() *não* deve ser submetida!