Curso de C++ para Resolução de Problemas

Projeto de Ensino e Aprendizagem de Programação para Olimpíadas

Terminal e Tipos de Entrada

Terminal

Em apoio ao software livre, neste material de referência trataremos apenas do bash (mais conhecido como shell), que é um interpretador de comandos entre o sistema operacional e o usuário, para compilar e executar os programas desenvolvidos em C++.

Compilando

O `g++` é compilador padrão da linguagem C++ disponível para sistemas operacionais baseados em UNIX, Linux e Mac OS X. Assim, faz parte do GNU Compiler Collection (GCC), que é um conjunto de compiladores de linguagens de programação produzido pelo Projeto GNU para construir um sistema operacional baseado em software livre.

```
g++ <nome_do_arquivo>.cpp
```

Para compilar seus programas utilizando o `g++` basta abrir o terminal e ir até o diretório no qual se encontra o arquivo do programa com a extensão `.cpp` e utilizar o formato acima como padrão, especificando o `<nome_do_arquivo>` do programa desejado, por exemplo `g++ codigo.cpp`. Dessa forma, como resultado teremos um arquivo chamado `a.out`.

```
g++ -o <nome_do_arquivo_executavel> <nome_do_arquivo>.cpp
```

Também é possível utilizar este outro formato, especificando o `<nome_do_arquivo>` do programa desejado e o `<nome_do_arquivo_executavel>` que será gerado a próxima etapa. Então, por exemplo, fazemos `g++ -o teste codigo.cpp`. Dessa forma, como resultado teremos um arquivo nomeado conforme a especificação. No caso do exemplo anterior, teríamos gerado um arquivo de nome `teste`.

- **Executando**

`./<nome_do_arquivo_executavel>`

O arquivo gerado na etapa de compilação (por exemplo, `teste`) pode ser executado diretamente utilizando o comando `./` do bash através do terminal.

`./teste < <arquivo_de_texto_entrada>`

Se quisermos alterar a entrada padrão do teclado para um `<arquivo_de_texto_entrada>` qualquer (por exemplo, `entrada.txt`) apenas utilizamos o especificador `<` para redirecionar o conteúdo do arquivo de texto especificado para a execução do programa. Isto é bastante útil quando queremos testar o programa desenvolvido para diversos casos de teste. Assim, apenas inserimos todos os casos de teste desejados em um arquivo de texto e verificamos se a saída produzida pelo programa é compatível com a saída esperada.

`./teste < entrada.txt > <arquivo_de_texto_saida>`

Também é possível redirecionar a saída produzida pelo programa para um `<arquivo_de_texto_saida>` qualquer (por exemplo, `saida.txt`) somente usando o especificador `>`. Isto pode ser útil na comparação de dois arquivos por meio do comando `diff` do bash, verificando se saída produzida pelo programa saida é igual a saída esperada. Por exemplo, `diff saida.txt resposta.txt` (se os arquivos possuírem exatamente o mesmo conteúdo a resposta gerada pela execução deste comando será vazia).

● Tipos de Entrada

Para exemplificar os tipos de entrada mais comuns para problemas de programação, usaremos como base uma situação-problema que requisita que seja desenvolvido um programa para somar dois números inteiros.

- **Um caso de teste**

```cpp
#include <iostream>

using namespace std;

int main(){
    int A, B;
    cin >> A >> B;
    cout << A+B << endl;
    return 0;
}
```

Neste cenário temos que o programa executa apenas um caso de teste por vez, somando dois números inteiros `A` e `B` e imprimindo a resposta `A+B` na saída padrão `cout`.

- **Muitos casos de teste**

```cpp
#include <iostream>

using namespace std;

int main(){
    int N, A, B;
    cin >> N;
    while(N--){
        cin >> A >> B;
        cout << A+B << endl;
    }
    return 0;
}
```

Neste outro cenário temos que o programa realiza uma quantidade positiva `N` de execuções do algoritmo desenvolvido, somando dois números inteiros `A` e `B` e imprimindo a resposta `A+B` na saída padrão `cout` para cada caso de teste lido da entrada padrão `cin`.

- **Vários casos de teste**

```cpp
#include <iostream>

using namespace std;

int main(){
    int A, B;
    while(cin >> A >> B){
        cout << A+B << endl;
    }
    return 0;
}
```

Por fim, quando não há especificação de uma quantidade exata de casos de teste para o problema, é necessário realizar a leitura da entrada padrão `cin` até que o procedimento `cin >> A >> B` retorne `false` (falso), assim garantindo que chegaremos até o final do arquivo (`EOF`) e que todos os casos de teste serão executados.

PROGRAMATHON

Projeto de Ensino e Aprendizagem de Programação para Olimpíadas

EQUIPE

Reginaldo M. Kuroshu (Coordenador Geral do Projeto)
Alexandre Hild Aono
Bruno Bernardo de Moura
Danilo Gustavo Hansen Laboissiere
Diogo Augusto Hansen Laboissiere
Lucas de Alencar Barbosa
Marcos Castro de Souza
Nathan de Melo Cruz
Rodrigo de Farias Ramires
Thauany Moedano
Victor de Sá Nunes
Willian da Silva Zocolau

APOIO

Pró-Reitoria de Extensão (PROEX-UNIFESP)

REALIZAÇÃO

Instituto de Ciência e Tecnologia (ICT-UNIFESP)

CONTATO

programathon.unifesp@gmail.com

PARA MAIS INFORMAÇÕES ACESSE

programathon-unifesp.github.io