

2024_1 - PROGRAMAÇÃO E DESENVOLVIMENTO DE SOFTWARE II - TN_TW - METATURMA

PAINEL > **MINHAS TURMAS** > **2024_1 - PROGRAMAÇÃO E DESENVOLVIMENTO DE SOFTWARE II - TN_TW - METATURMA** > **GERAL**

> **VPL3 - TADS: CONTROLE BANCÁRIO (4,0 PTS) - ENTREGA EM 04/04 ATÉ AS 14:00**


 Descrição

 Lista de envios

 Similaridades

 Atividade de teste

VPL3 - TADs: Controle Bancário (4,0 pts) - Entrega em 04/04 até as 14:00




 **Data de entrega:** quinta, 4 Abr 2024, 14:00

 **Arquivos requeridos:** ContaBancaria.hpp, ContaBancaria.cpp, Banco.hpp, Banco.cpp, main.cpp ( [Baixar](#))

Tipo de trabalho:  Trabalho individual

 **Configurações de notas:** Nota máxima: 4

 **Executar:** Sim. **Script de execução:** CPP-11.  **Depurar:** Sim. **Script de depuração:** CPP.  **Avaliar:** Sim

Atribuição automática de nota: Sim.  **Tempo máximo de execução:** 120 s.  **Memória máxima utilizada:** 512 MiB.  **Tamanho máximo de arquivos de execução:** 128 MiB.

O Objetivo desse projeto é praticar os conceitos de Tipos Abstratos de Dados (TAD) e Modularização.

Você foi contratado por um pequeno banco para implementar um sistema de forma a automatizar as suas operações. Basicamente os clientes podem criar contas, depositar, sacar e fazer transferências (pix). Para implementar o seu sistema você vai criar 2 TADs e um arquivo main:

TAD Conta Bancária

O TAD Conta Bancária armazena os dados da conta: id (inteiro), cliente (string) e saldo (float) e possui as seguintes operações:

- Construtor: inicializa uma nova conta com um id e cliente passados como parâmetro e saldo igual a 0.
- Depósito: recebe um valor como parâmetro e adiciona ao saldo da conta.
- Saque: recebe um valor como parâmetro e subtrai do saldo da conta.
- Pix: recebe como parâmetros um apontador para uma outra conta e um valor, e transfere o valor da conta que está chamando o método para a conta passada como parâmetro.
- Imprime: imprime em uma linha o id, nome do cliente e saldo, separados por espaço e com um `endl` no final. O saldo deve ser impresso com 2 casas decimais.

TAD Banco

O TAD banco é usada para armazenar as contas bancárias. Basicamente, ele vai ter como atributos um número inteiro que armazena o número total de contas e um vetor com apontadores para contas. (Para simplificar, você pode considerar que o número máximo de contas do Banco é 20 e usar um vetor simples com alocação estática). O TAD Banco possui as seguintes operações:

- Construtor: inicializa o TAD colocando o valor 0 no número de contas e inicializando todos elementos do vetor de contas com `nullptr`.
- CriaConta: cria uma nova ContaBancária com id e nome passados como parâmetro e a coloca no vetor de contas, incrementando também o número de contas. Note que não podem haver contas com ids repetidos, portanto o seu método deve retornar um apontador para a conta criada ou, em caso de falhas, `nullptr`.
- Pesquisa: recebe um id de uma conta e faz uma pesquisa por uma conta com esse id no vetor de contas, retornando um apontador para a conta encontrada ou `nullptr` caso a conta não exista.
- ListaContas: imprime as informações de todas as contas cadastradas no Banco (id, cliente, saldo) uma conta por linha.

Main:



O seu programa principal faz a gerência das operações do banco. Ele vai ser um basicamente um loop que lê comandos da entrada (C, D, S, P, I, T) juntamente com os parâmetros e faz as operações de acordo. Para cada comando, o sistema imprime uma mensagem com uma quebra de linha (`endl`) no final. Importante: as mensagens devem ser exatamente como especificado abaixo para evitar erros na correção automática. As operações são:

C `<id>` `<nome>`: cria uma conta com id e nome indicados. Deve ser impresso: "conta criada" em caso de sucesso ou "ERRO: conta repetida" em caso de falha.

D `<id>` `<valor>`: faz um depósito do valor na conta indicada. Você poderá considerar que os valores fornecidos serão sempre positivos. Deve ser impresso: "deposito efetuado" em caso de sucesso ou "ERRO: conta inexistente" caso o id da conta seja inválido.

S `<id>` `<valor>`: faz um saque do valor na conta indicada. Você poderá considerar que os valores fornecidos serão sempre positivos. Deve ser impresso: "saque efetuado" em caso de sucesso ou "ERRO: conta inexistente" caso o id da conta seja inválido.

P `<id>` `<dest>` `<valor>`: faz um pix, transferindo o valor da conta indicada por `<id>` para a conta indicada por `<dest>`. Você poderá considerar que os valores fornecidos serão sempre positivos. Deve ser impresso: "pix efetuado" em caso de sucesso ou "ERRO: conta inexistente" caso as uma das contas indicadas sejam inválidas.

I: lista as informações de todas as contas cadastradas no Banco (id, cliente, saldo) uma conta por linha.

T: termina a execução do programa.

Segue abaixo um exemplo de uma execução do Sistema:

```
C 123 Luiz
conta criada
C 123 Maria
ERRO: conta repetida
C 999 Maria
conta criada
D 123 1000
deposito efetuado
S 555 500
ERRO: conta inexistente
S 123 200
saque efetuado
P 123 999 300
pix efetuado
I
123 Luiz 500.00
999 Maria 300.00
T
```

Você deverá submeter 5 arquivos: `ContaBancaria.hpp`, `ContaBancaria.cpp`, `Banco.hpp`, `Banco.cpp` e `main.cpp`. No seu programa principal, você não deverá acessar diretamente os atributos dos seus TADs. Todo o acesso deverá ser feito através dos métodos implementados.

Arquivos de execução

`vpl_run.sh`

```
1 #!/bin/bash
2 # $Id: cpp_run.sh,v 1.3 2012-07-25 19:02:21 juanca Exp $
3 # Default C++ language run script for VPL
4 # Copyright (C) 2012 Juan Carlos Rodríguez-del-Pino
5 # License http://www.gnu.org/copyleft/gpl.html GNU GPL v3 or later
6 # Author Juan Carlos Rodríguez-del-Pino <jcrodriguez@dis.ulpgc.es>
7
8 #load common script and check programs
9 . common_script.sh
10 check_program g++
11 get_source_files cpp hpp C
12
13 #echo $SOURCE_FILES
14
15 VAZIO="0"
16 #echo $VAZIO
17
18 for FILENAME in $SOURCE_FILES
19 do
20     if ! [ -s "$FILENAME" ] ; then
21         echo "NÃO PODEM SER SUBMETIDOS ARQUIVOS VAZIOS! ($FILENAME)"
22         VAZIO="1"
23     fi
24 done
25
26 if [ $VAZIO -eq "1" ] ; then
27     exit 1
28 fi
29
30 #exit 0;
31
32 #compile
33 g++ -std=c++11 -Wall -fno-diagnostics-color -o vpl_execution $SOURCE_FILES -lm -lutil
```

vpl_debug.sh

```
1 #!/bin/bash
2 # $Id: cpp_debug.sh,v 1.4 2012-09-24 15:13:21 juanca Exp $
3 # Default C++ language debug script for VPL
4 # Copyright (C) 2014 Juan Carlos Rodríguez-del-Pino
5 # License http://www.gnu.org/copyleft/gpl.html GNU GPL v3 or later
6 # Author Juan Carlos Rodríguez-del-Pino <jcrodriguez@dis.ulpgc.es>
7
8 #load common script and check programs
9 . common_script.sh
10 check_program g++
11 get_source_files cpp C
12
13 #compile
14 g++ -o vpl_program -g -O0 -std=c++11 -Wall -fdiagnostics-color=never $SOURCE_FILES -lm -lutil
15
16 valgrind --leak-check=full --show-leak-kinds=all --track-origins=yes ./vpl_program
17
18 if [ -f vpl_program ] ; then
19     cat common_script.sh > vpl_execution
20     chmod +x vpl_execution
21     if [ "$(command -v ddd)" == "" ] ; then
22         check_program gdb
23         echo "gdb vpl_program" >> vpl_execution
24     else
25         echo "ddd --quiet --gdb -geometry 800x600 vpl_program &>/dev/null" >> vpl_execution
26         mkdir .ddd
27         mkdir .ddd/sessions
28         mkdir .ddd/themes
29         cat >.ddd/init <<END_OF_FILE
30 Ddd*splashScreen: off
31 Ddd*startupTips: off
32 Ddd*suppressWarnings: on
33 Ddd*displayLineNumbers: on
34 Ddd*saveHistoryOnExit: off
35
36 ! DO NOT ADD ANYTHING BELOW THIS LINE -- DDD WILL OVERWRITE IT
37 END_OF_FILE
38     mv vpl_execution vpl_wexecution
39 fi
40 fi
```

vpl_evaluate.cases

```
1 Case = Testes de Consistência
2 input =
3 C 123 Luiz
4 C 123 Maria
5 D 555 1000
6 S 555 500
7 P 123 456 1000
8 P 456 123 1000
9 P 22 23 1000
10 I
11 T
12 output =
13 conta criada
14 ERRO: conta repetida
15 ERRO: conta inexistente
16 ERRO: conta inexistente
17 ERRO: conta inexistente
18 ERRO: conta inexistente
19 ERRO: conta inexistente
20 123 Luiz 0.00
21
22 Case = Testes Simples
23 input =
24 C 123 Luiz
25 D 123 1000
26 S 123 500
27 S 123 300
28 I
29 T
30 output =
31 conta criada
32 deposito efetuado
33 saque efetuado
34 saque efetuado
35 123 Luiz 200.00
36
37 Case = 2 Contas
38 input =
39 C 123 Luiz
40 C 123 Maria
41 C 999 Maria
42 D 123 1000
43 S 555 500
44 S 123 200
45 P 123 999 300
46 I
47 T
48 output =
49 conta criada
50 ERRO: conta repetida
51 conta criada
52 deposito efetuado
53 ERRO: conta inexistente
54 saque efetuado
55 pix efetuado
56 123 Luiz 500.00
57 999 Maria 300.00
58
59 Case = 4 Contas
60 input =
61 C 123 Luiz
62 C 222 Marcos
63 D 222 1000
64 P 222 123 200
65 C 707 Maria
66 P 123 707 200
67 C 999 Julia
68 D 999 5000
69 S 999 2000
70 S 999 1500
71 D 999 1000
72 P 999 222 1000
73 P 222 707 1000
74 I
75 T
76 output =
77 conta criada
78 conta criada
79 deposito efetuado
80 pix efetuado
81 conta criada
82 pix efetuado
83 conta criada
84 deposito efetuado
85 saque efetuado
86 saque efetuado
87 deposito efetuado
88 pix efetuado
89 pix efetuado
90 123 Luiz 0.00
91 222 Marcos 800.00
92 707 Maria 1200.00
93 999 Julia 1500.00
94
```

[VPL](#)

