Programação Orientada a Objetos

Uma das principais limitações da programação procedural é que não existe uma forma simples de criar uma conexão forte entre os dados e funções.

No código seguinte, os dados são globais e as todas as funções acessam diretamente os dados. Não existe nenhuma conexão entre os dados e as funções que manipulam esses dados

```
#include <stdio.h>
#include <string.h>

char nomeCorrentista[20];
double saldo;
bool contaEhEspecial;

void abreContaSimples(char *nome){
    strcpy(nomeCorrentista, nome);
    saldo = 0;
    contaEhEspecial = false;
}

void abreConta(char * nome, double deposito, bool ehEspecial){
    strcpy(nomeCorrentista, nome);
    saldo = deposito;
    contaEhEspecial = ehEspecial;
}

void deposita(int valor){
    saldo += valor;
}


void retira(int valor){
    if( contaEhEspecial ){
        saldo -= valor;
    }else{
        if( valor <= saldo ){
            saldo -= valor;
        }else{
            printf("Valor insuficiente");
        }
    }
}


void mostra(){
```

```
    printf("Nome: %s\n", nomeCorrentista);
    printf("Saldo: %lf\n", saldo);
    printf("Especial: %s\n", (contaEhEspecial? "sim" : "nao"));
}

int main(){
    abreContaSimples((char*)"Wladimir");
    deposita(5000);
    mostra();
}
```

No código seguinte, criamos uma `struct` para agrupar os dados relacionados com uma conta bancária e todas as funções que manipulam os dados de uma conta bancária recebem como primeiro parâmetro um ponteiro para uma conta bancária. A passagem do ponteiro para a conta bancária permite que os dados relacionado com a conta bancária possam ser acessados e alterados.

```
#include <stdio.h>
#include <string.h>

typedef struct{
    char nomeCorrentista[20];
    double saldo;
    bool contaEhEspecial;
} ContaBancaria;

void abreContaSimples(ContaBancaria* c, char *nome){
    strcpy(c->nomeCorrentista, nome);
    c->saldo = 0;
    c->contaEhEspecial = false;
}

void abreConta(ContaBancaria * c, char * nome, double deposito, bool ehEspecial){
    strcpy(c->nomeCorrentista, nome);
    c->saldo = deposito;
    c->contaEhEspecial = ehEspecial;
}

void deposita(ContaBancaria * c, int valor){
    c->saldo += valor;
}


void retira(ContaBancaria * c, int valor){
    if( c->contaEhEspecial ){
        c->saldo -= valor;
    }else{
```

```
        if( valor <= c->saldo ){
            c->saldo -= valor;
        }else{
            printf("Valor insuficiente");
        }
    }
}


void mostra(const ContaBancaria * c){
    printf("Nome: %s\n", c->nomeCorrentista);
    printf("Saldo: %lf\n", c->saldo);
    printf("Especial: %s\n", (c->contaEhEspecial? "sim" : "nao"));
}

int main(){

    ContaBancaria c1;
    ContaBancaria c2;

    abreContaSimples(&c1, (char*)"Wladimir");
    deposita(&c1, 5000);
    mostra(&c1);

    abreConta(&c2, (char*)"Andre", 2000, true);
    deposita(&c2, 10000);
    c2.saldo += 1000;
    mostra(&c2);

}
```

No código anterior, a struct Conta Bancaria é passada por referência para as funções. A passagem por referência pode ser implementada de duas maneiras:
* O subprograma recebe um endereço de acesso através de um ponteiro. Esse ponteiro permite acessar/alterar o parâmetro real passado para o subprograma.
* O subprograma tem a permissão de acessar o parâmetro a partir da criação de um apelido para a variável. A partir do apelido, o parâmetro real pode ser alterado no subprograma.

No código seguinte, mostramos a implementação do código anterior utilizando a segunda estratégia de implementação da passagem por referência.

```
#include <iostream>
#include <string>

using namespace std;
```

```
typedef struct{
    string nomeCorrentista;
    double saldo;
    bool contaEhEspecial;
} ContaBancaria;

void abreContaSimples(ContaBancaria & c, string nome){
    c.nomeCorrentista =  nome;
    c.saldo = 0;
    c.contaEhEspecial = false;
}

void abreConta(ContaBancaria & c, string nome, double deposito, bool ehEspecial){
    c.nomeCorrentista =  nome;
    c.saldo = deposito;
    c.contaEhEspecial = ehEspecial;
}

void deposita(ContaBancaria & c, int valor){
    c.saldo += valor;
}


void retira(ContaBancaria & c, int valor){
    if( c.contaEhEspecial ){
        c.saldo -= valor;
    }else{
        if( valor <= c.saldo ){
            c.saldo -= valor;
        }else{
            cout << "Valor insuficiente" << endl;
        }
    }
}


void mostra(const ContaBancaria & c){
    cout << "Nome: " <<  c.nomeCorrentista << endl;
    cout << "Saldo: " <<  c.saldo << endl;
    cout << "Especial: "<<  (c.contaEhEspecial? "sim" : "nao") << endl;
}

int main(){
```

```
    ContaBancaria c1;
    ContaBancaria c2;

    abreContaSimples(c1, "Wladimir");
    deposita(c1, 5000);
    mostra(c1);

    abreConta(c2, "Andre", 2000, true);
    deposita(c2, 10000);
    c2.saldo += 1000;
    mostra(c2);

}
```

A linguagem C++ permite colocar dados e funções dentro da struct. Note que existe algumas vantagens de agrupar os dados e funções. Agora, os dados não precisam ser passado como referência para as funções. Como as funções e os dados estão juntos, todas as variáveis da struct são visíveis nas funções. Contudo, no exemplo abaixo, podemos alterar o saldo sem utilizar as funções disponibilizadas pela struct.

```
#include <iostream>
#include <string>

using namespace std;


typedef struct{
    string nomeCorrentista;
    double saldo;
    bool contaEhEspecial;

    void abreContaSimples(string nome){
        nomeCorrentista =  nome;
        saldo = 0;
        contaEhEspecial = false;
    }

    void abreConta(string nome, double deposito, bool ehEspecial){
        nomeCorrentista =  nome;
        saldo = deposito;
        contaEhEspecial = ehEspecial;
    }

    void deposita(int valor){
```

```
        saldo += valor;
    }


    void retira(int valor){
        if( contaEhEspecial ){
            saldo -= valor;
        }else{
            if( valor <= saldo ){
                saldo -= valor;
            }else{
                cout << "Valor insuficiente" << endl;
            }
        }
    }


    void mostra(){
        cout << "Nome: " <<  nomeCorrentista << endl;
        cout << "Saldo: " <<  saldo << endl;
        cout << "Especial: "<<  (contaEhEspecial? "sim" : "nao") << endl;
    }

} ContaBancaria;


int main(){

    ContaBancaria c1;
    ContaBancaria c2;
    ContaBancaria c3{ "Fabio", 5000, false};

    c1.abreContaSimples("Wladimir");
    c1.deposita(5000);
    c1.mostra();

    c2.abreConta("Andre", 2000, true);
    c2.deposita(10000);
    c2.saldo += 1000;
    c2.mostra();

    c3.mostra();

}
```

Na cadeira de POO, iremos aprender o conceito de classe que estende o conceito

das struct da linguagem C++. Com as classes, poderemos construir uma conexão ainda mais forte dos dados com as funções.