Sistema De Pagamento

AUTHOR
Versão 2.0
06/22/2022

Sumário

Table of contents

Índice Hierárquico

Hierarquia de Classes

Esta lista de hierarquias está parcialmente ordenada (ordem alfabética):

Índice dos Componentes

Lista de Classes

Aqui estão as classes, estruturas, uniões e interfaces e suas respectivas descrições:

Índice dos Arquivos

Lista de Arquivos

Esta é a lista de todos os arquivos e suas respectivas descrições:

Classes

Referência da Classe Diretor

Diagrama de hierarquia para Diretor:

IMAGE

Membros Públicos

Diretor ()
Diretor (string areaSupervisao, string areaFormacao, string **nome**, string **endereco**, string **telefone**, string **designacao**, tm **dataIngresso**, double **salario**)
virtual **~Diretor** ()
void **setAreaSupervisao** (string areaSupervisao)
void **setAreaFormacao** (string areaFormacao)
string **getAreaSupervisao** ()
string **getAreaFormacao** ()
void **aumentoSalarial** ()
Método aumentoSalarial aumenta o salario do diretor em 20%.

Outros membros herdados

Construtores e Destrutores

Diretor::Diretor ()

```
333                         : Funcionario(3){
334
335 }
```

Diretor::Diretor (string *areaSupervisao*, string *areaFormacao*, string *nome*, string *endereco*, string *telefone*, string *designacao*, tm *dataIngresso*, double *salario*)

```
337       : Funcionario(nome, endereco, telefone, designacao, dataIngresso,
salario){
338       this->areaSupervisao = areaSupervisao;
339       this->areaFormacao = areaFormacao;
340 }
```

virtual Diretor::~Diretor () `[inline], [virtual]`

```
318 {}
```

Funções membros

void Diretor::aumentoSalarial () `[virtual]`

Método aumentoSalarial aumenta o salario do diretor em 20%.

Implementa **Funcionario** (*p.13*).

```
355                                     {
356       this->salario = this->salario + (salario*(20/100));
357       this->tributarSalario();
358 }
```

string Diretor::getAreaFormacao ()

```
350                                       {
351       return this->areaFormacao;
```

```
352 }
```

string Diretor::getAreaSupervisao ()

```
347                                          {
348       return this->areaSupervisao;
349 }
```

void Diretor::setAreaFormacao (string *areaFormacao*)

```
344                                                          {
345       this->areaFormacao = areaFormacao;
346 }
```

void Diretor::setAreaSupervisao (string *areaSupervisao*)

```
341                                                                  {
342       this->areaSupervisao = areaSupervisao;
343 }
```

A documentação para essa classe foi gerada a partir do seguinte arquivo:

codigo/**main (5).cpp**

Referência da Classe Empresa

Membros Públicos

Empresa ()
~Empresa ()
int **getVectorSize** ()

O método getVectorSize retorna o tamanho do vetor de funcionarios.

Funcionario * get_Func_com_index (int)
vector< double > **empresa_anual** ()
double **empresa_mensal** (int)
vector< double > **funcionario_anual** (int)
double **funcionario_mensal** (int, int)
vector< int > **funcionarios_achados** (vector< string >)
bool **compare_datas** (tm, tm)
void **add_func** (**Funcionario** *)

O método add_func Adiciona um funcionario ao vector de funcionarios.

void **att_func** (**Funcionario** *, int)
void **aumentoSalarioGeral** ()
void **apagar_funcionario** (int)

Construtores e Destrutores

Empresa::Empresa ()

```
434                         {
435
436       for(int i = 0; i < 12; i++){
437            this->folhaSalarial[i] = 0.0;
438       }
439 }
```

Empresa::~Empresa () `[inline]`

```
418 {}
```

Funções membros

void Empresa::add_func (Funcionario * *fun_*)

O método add_func Adiciona um funcionario ao vector de funcionarios.

```
678                                                    {
679       this->funcionarios.push_back(fun_);
680 }
```

void Empresa::apagar_funcionario (int *index*)

O método apagar_funcionario deleta o funcionario no indice passado como parametro e automaticamente reorganiza o vector de funcionarios.

```
706                                                      {
707       this->funcionarios.erase(funcionarios.begin() + index);
708 }
```

void Empresa::att_func (Funcionario * *fun*, int *index*)

O método att_func atualiza o ponteiro de funcionario de indice passado como parametro para o ponteiro de funcionario tambem passado tambem como parametro.

```
686                                                                    {
687       funcionarios[index] = fun;
688 }
```

void Empresa::aumentoSalarioGeral ()

O método aumentoSalarioGeral Raliza o aumento salarial em todos os funcionarios nas seguintes proporções: **Operador** - 5% **Gerente** - 10% **Diretor** - 15% **Presidente** - 20%

```
698                                                {
699       for(int i = 0; i < funcionarios.size(); i++){
700            funcionarios[i]->aumentoSalarial();
701       }
702 }
```

bool Empresa::compare_datas (tm *data_inicio*, tm *data_final*)

O método compare_datas retorna true caso as datas passadas como parametro forem iguais e false caso elas diferenciem, nao sao considerados horarios.

```
665                                                                                {
666       if(data_inicio.tm_mday == data_final.tm_mday && data_inicio.tm_mon ==
667            data_final.tm_mon && data_inicio.tm_year == data_final.tm_year){
668            return true;
669       }
670       else{
671            return false;
672       }
673 }
```

vector< double > Empresa::empresa_anual ()

O método empresa_anual retorna um vector do tipo double com os valores mensais da folha salarial em cada mes ate o indice 11 e, como ultimo parametro, o total anual somado.

```
491                                                    {
492       vector<double> valores;
493       double mensal, total = 0.0;
494
495       for(int i = 0; i < 12; i++){
496            mensal = empresa_mensal(i);
497            total += mensal;
498
499            valores.push_back(mensal);
500       }
501       valores.push_back(total);
502
503       return valores;
504 }
```

double Empresa::empresa_mensal (int *index*)

O método empresa_mensal retorna um double com o valor da folha salarial do mes passado como parametro de 0 a 11.

```
510                                                      {
511
512       bool notExists = (this->folhaSalarial[index] == 0.0);
513       if(notExists){
514            for(int i = 0; i < funcionarios.size(); i++){
515                 this->folhaSalarial[index]+=funcionarios[i]-
>get_SalarioMes(index);
516            }
517       }
518
519       return this->folhaSalarial[index];
520 }
```

vector< double > Empresa::funcionario_anual (int *index*)

O método funcionario_anual retorna um vector do tipo double com os valores mensais da folha salarial em cada mes ate o indice 11 e, como ultimo parametro, o total anual somado.

```
527                                                                 {
528       vector<double> valores;
529       double mensal, total = 0.0;
530
531       for(int i = 0; i < 12; i++){
532            mensal = funcionario_mensal(index, i);
533            total += mensal;
534
535            valores.push_back(mensal);
536       }
537       valores.push_back(total);
538
539       return valores;
540
541 }
```

double Empresa::funcionario_mensal (int *index_func*, int *index_mes*)

O método empresa_mensal retorna um double com o valor da folha salarial do mes passado como parametro de 0 a 11.

```
547
{
548       return this->funcionarios[index_func]->get_SalarioMes(index_mes);
549 }
```

vector< int > Empresa::funcionarios_achados (vector< string > *parametros*)

O método funcionarios_achados retorna um vector de valores inteiros que representara o indice dos funcionarios achados de acordo com os parametros passados em um vector string q foi tratado no botao de busca, formulario edit.

```
556
{
557       vector<bool> existe;
558       vector<int> achados;
559       bool achou_um = false;
560
561       for(const string &str : parametros){
562            if(str.compare("")) existe.push_back(true);
563            else existe.push_back(false);
564       }
565
566       bool todos_vazios = true;
567       for(bool b : existe){
568            if(b) todos_vazios = false;
569            achou_um = true;
570       }
571
572       if(todos_vazios){
573            for(int i = 0; i < funcionarios.size(); i++){
574                 achados.push_back(i);
575            }
576       }else{
577            bool igual;
578            for(int i = 0; i < funcionarios.size(); i++){
579                 igual = false;
580
581                 //nome
582                 if(existe[0]){
583                      if(funcionarios[i]->getNome().compare(parametros[0])){
584                           igual = false;
585                      }else{
586                           igual = true;
587                      }
588                 }
589                 //endereco
590                 if(existe[1]){
591                      if(funcionarios[i]->getEndereco().compare(parametros[1])){
```

```
592                         igual = false;
593                    }else{
594                         igual = true;
595                    }
596               }
597
598               //codigo
599               if(existe[2]){
600                    if(funcionarios[i]-
>getCodFuncionario().compare(parametros[2])){
601                         igual = false;
602                    }else{
603                         igual = true;
604                    }
605               }
606
607               //designacao
608               if(existe[3]){
609                    if(funcionarios[i]-
>getDesignacao().compare(parametros[3])){
610                         igual = false;
611                    }else{
612                         igual = true;
613                    }
614               }
615
616               //data
617               if((existe[4] && existe[5] && existe[6]) && (existe[7]
618                         && existe[8] && existe[9])){
619                    if((stoi(parametros[4]) > 28 && stoi(parametros[5]) ==
02 ) ||
620                         (stoi(parametros[7]) > 28 && stoi(parametros[8]) ==
02 )){
621                         throw ("Fevereiro não pode ter mais que 28 dias");
622                    }
623
624                    tm data_inicio, data_fim;
625
626                    data_inicio.tm_mday = stoi(parametros[4]);
627                    data_inicio.tm_mon = stoi(parametros[5]);
628                    data_inicio.tm_year = stoi(parametros[6]);
629
630                    data_fim.tm_mday = stoi(parametros[7]);
631                    data_fim.tm_mon = stoi(parametros[8]);
632                    data_fim.tm_year = stoi(parametros[9]);
633
634                    igual = false;
635                    while(!compare_datas(data_inicio, data_fim)){
636                         if(funcionarios[i]->ComparaDatas(data_inicio)){
637                              igual = true;
638                              break;
639                         }
640                         avancarDia(data_inicio);
641                    }
642               }
643
644               if(igual){
645                    achados.push_back(i);
646                    achou_um = true;
647               }
648          }
649     }
650
651     if(!achou_um){
652          throw ("Não foi encontrado um funcionário com estes parametros");
653     }
654     else{
655          return achados;
656     }
657
658     existe.clear();
659     achados.clear();
660 }
```

Funcionario * Empresa::get_Func_com_index (int *index*)

O método get_Func_com_index retorna o **Funcionario** do vetor de funcionarios que tenha o indice passado como parametro.

```
483                                                          {
484       return this->funcionarios[index];
485 }
```

int Empresa::getVectorSize ()

O método getVectorSize retorna o tamanho do vetor de funcionarios.

```
476                                  {
477       return funcionarios.size();
478 }
```

A documentação para essa classe foi gerada a partir do seguinte arquivo:

codigo/**main (5).cpp**

Referência da Classe Funcionario

Diagrama de hierarquia para Funcionario:

IMAGE

Membros Públicos

Funcionario (int=0)
Funcionario (string **nome**, string **endereco**, string **telefone**, string **designacao**, tm **dataIngresso**, double **salario**)
virtual **~Funcionario** ()
void **setCodFuncionario** (string CodFuncionario)
void **setNome** (string **nome**)
void **setEndereco** (string **endereco**)
void **setTelefone** (string **telefone**)
void **setDesignacao** (string **designacao**)
void **setDataIngresso** (tm **dataIngresso**)
void **setSalario** (double **salario**)
void **setSalario_tributado** (double newSalario_tributado)
void **setValor_hora** (double newValor_hora)
string **getCodFuncionario** ()
string **getNome** ()
string **getEndereco** ()
string **getTelefone** ()
string **getDesignacao** ()
tm **getDataIngresso** ()
double **getSalario** ()
double **getSalario_tributado** () const
double **getValor_hora** () const
bool **ComparaDatas** (tm data)
void **tributarSalario** ()
double **get_SalarioMes** (int index)
int **getDiasTrabalhados** (int index)
int **getHorasExtras** (int index)
virtual void **aumentoSalarial** ()=0

Atributos Protegidos

string **codFuncionario**
string **nome**
string **endereco**
string **telefone**
string **designacao**
tm **dataIngresso**
double **salario**
double **salario_tributado**
double **valor_hora**
int **horasTrabalhadas** [12]
int **diasTrabalhados** [12]
double **salariosMensais** [12]

Construtores e Destrutores

Funcionario::Funcionario (int *t* = 0)

Construtor da classe **Funcionario** recebe um inteiro e inicilializa todas as classes filhas.

```
66                                          {
67       switch(t){
68            case 0:
69                 this->designacao = "undefined";
```

```
70
71                  break;
72            case 1:
73                  this->designacao = "Operador";
74
75                  break;
76            case 2:
77                  this->designacao = "Gerente";
78
79                  break;
80            case 3:
81                  this->designacao = "Diretor";
82
83                  break;
84            case 4:
85                  this->designacao = "Presidente";
86
87                  break;
88            default:
89                  break;
90      }
91 }
```

Funcionario::Funcionario (string *nome*, string *endereco*, string *telefone*, string *designacao*, tm *dataIngresso*, double *salario*) [explicit]

Construtor inicilializa todos os atributos da classe e calcula o valor do salario mensal.

```
95
{
96      this->nome = nome;
97      this->endereco = endereco;
98      this->telefone = telefone;
99      this->designacao = designacao;
100      this->dataIngresso.tm_mday = dataIngresso.tm_mday;
101      this->dataIngresso.tm_mon = dataIngresso.tm_mon;
102      this->dataIngresso.tm_year = dataIngresso.tm_year;
103      this->salario = salario;
104      tributarSalario();
105
106      double salario_mensal;
107      valor_hora = (salario_tributado / 25) / 24;
108
109      for(int i = 0; i < 12; i++){
110           horasTrabalhadas[i] = rand() % 20;
111           diasTrabalhados[i] = rand() % 25;
112
113           salario_mensal = diasTrabalhados[i] * (valor_hora * 8)
114                                + horasTrabalhadas[i] * (valor_hora * 1.5);
115
116           salariosMensais[i] = salario_mensal;
117      }
118
119      int cod = (rand() % 1000 + 1000);
120      this->codFuncionario = to_string(cod);
121 }
```

virtual Funcionario::~Funcionario () [inline], [virtual]

```
18 {}
```

Funções membros

virtual void Funcionario::aumentoSalarial () [pure virtual]

Implementado por **Operador** (*p.20*), **Gerente** (*p.18*), **Diretor** (*p.5*) e **Presidente** (*p.21*).

bool Funcionario::ComparaDatas (tm *data*)

O método ComparaDatas recebe uma struct tm com uma data e compara com a data de ingresso do **Funcionario** e retorna igual ou diferente.

```
175                                                 {
176       if(this->dataIngresso.tm_mday == data.tm_mday &&
177            this->dataIngresso.tm_mon == data.tm_mon &&
178            this->dataIngresso.tm_year == data.tm_year){
179            return true;
180       }else{
181            return false;
182       }
183 }
```

double Funcionario::get_SalarioMes (int *index*)

```
147                                                       {
148       return salariosMensais[index];
149 }
```

string Funcionario::getCodFuncionario ()

```
150                                                 {
151       return this->codFuncionario;
152 }
```

tm Funcionario::getDataIngresso ()

```
165                                           {
166       return this->dataIngresso;
167 }
```

string Funcionario::getDesignacao ()

```
162                                             {
163       return this->designacao;
164 }
```

int Funcionario::getDiasTrabalhados (int *index*)

Método getDiasTrabalhados recebe um index que representa o mês e retorna os dias trabalhados naquele mês.

```
227                                                        {
228       return diasTrabalhados[index];
229 }
```

string Funcionario::getEndereco ()

```
156                                           {
157       return this->endereco;
158 }
```

int Funcionario::getHorasExtras (int *index*)

Método getHorasExtras recebe um index que representa o mês e retorna as horas extras trabalhados naquele mês.

```
234                                                   {
235       return horasTrabalhadas[index];
236 }
```

string Funcionario::getNome ()

```
153                                        {
154       return this->nome;
155 }
```

double Funcionario::getSalario ()

```
168                                          {
169       return this->salario;
170 }
```

double Funcionario::getSalario_tributado () const

```
243                                                          {
244       return salario_tributado;
245 }
```

string Funcionario::getTelefone ()

```
159                                         {
160       return this->telefone;
161 }
```

double Funcionario::getValor_hora () const

```
237                                                 {
238       return valor_hora;
239 }
```

void Funcionario::setCodFuncionario (string *CodFuncionario*)

```
123                                                                                 {
124       this->codFuncionario = CodFuncionario;
125 }
```

void Funcionario::setDataIngresso (tm *dataIngresso*)

```
138                                                                 {
139       this->dataIngresso.tm_mday = dataIngresso.tm_mday;
140       this->dataIngresso.tm_mon = dataIngresso.tm_mon;
141       this->dataIngresso.tm_year = dataIngresso.tm_year;
142 }
```

void Funcionario::setDesignacao (string *designacao*)

```
135                                                                 {
136       this->designacao = designacao;
137 }
```

void Funcionario::setEndereco (string *endereco*)

```
129                                                           {
130       this->endereco = endereco;
131 }
```

void Funcionario::setNome (string *nome*)

```
126                                                 {
127       this->nome = nome;
128 }
```

void Funcionario::setSalario (double *salario*)

```
143                                                       {
144       this->salario = salario;
145       tributarSalario();
146 }
```

void Funcionario::setSalario_tributado (double *newSalario_tributado*)

```
246
{
247       salario_tributado = newSalario_tributado;
248 }
```

void Funcionario::setTelefone (string *telefone*)

```
132                                                           {
133       this->telefone = telefone;
134 }
```

void Funcionario::setValor_hora (double *newValor_hora*)

```
240                                                                     {
241       valor_hora = newValor_hora;
242 }
```

void Funcionario::tributarSalario ()

O método tributarSalario desconta o imposto de renda e a previdência do salario bruto.

```
187                                         {
188
189       double parte_previdencia = 0.0;
190       double parte_IR;
```

```
191
192       if(salario < 1212){
193            parte_previdencia = salario * (7.5/100);
194
195       }else if(salario >= 1212.01 && salario <= 2427.35){
196            parte_previdencia = salario * (9.0/100);
197
198       }else if(salario >= 2427.36 && salario <= 3641.03){
199            parte_previdencia = salario * (12.0/1000);
200
201       }else{
202            parte_previdencia = salario * (14.0/100);
203       }
204
205       if(salario < 1903.98){
206            parte_IR = 0;
207       }
208       else if(salario > 1903.99 && salario < 2826.65){
209            parte_IR = salario * (7.5/100);
210       }
211       else if(salario > 2826.66 && salario < 3751.05){
212            parte_IR = salario * (15/100);
213       }
214       else if(salario > 3751.06 && salario < 4664.68){
215            parte_IR = salario * (22.5/100);
216       }
217       else{
218            parte_IR = salario * (27.5/100);
219       }
220
221       salario_tributado = salario - parte_previdencia - parte_IR;
222 }
```

Atributos

string Funcionario::codFuncionario `[protected]`

tm Funcionario::dataIngresso `[protected]`

string Funcionario::designacao `[protected]`

int Funcionario::diasTrabalhados[12] `[protected]`

string Funcionario::endereco `[protected]`

int Funcionario::horasTrabalhadas[12] `[protected]`

string Funcionario::nome `[protected]`

double Funcionario::salario `[protected]`

double Funcionario::salario_tributado `[protected]`

double Funcionario::salariosMensais[12] `[protected]`

string Funcionario::telefone `[protected]`

double Funcionario::valor_hora `[protected]`

A documentação para essa classe foi gerada a partir do seguinte arquivo:

codigo/**main (5).cpp**

Referência da Classe Gerente

Diagrama de hierarquia para Gerente:

IMAGE

Membros Públicos

Gerente ()
Gerente (string areaSupervisao, string **nome**, string **endereco**, string **telefone**, string **designacao**, tm **dataIngresso**, double **salario**)
virtual **~Gerente** ()
void **setAreaSupervisao** (string AreaSupervisao)
string **getAreaSupervisao** ()
void **aumentoSalarial** ()
Método aumentoSalarial aumenta o salario do gerente em 10%.

Outros membros herdados

Construtores e Destrutores

Gerente::Gerente ()

```
292                         : Funcionario(2){
293
294 }
```

Gerente::Gerente (string *areaSupervisao*, string *nome*, string *endereco*, string *telefone*, string *designacao*, tm *dataIngresso*, double *salario*)

```
296       : Funcionario(nome, endereco, telefone, designacao, dataIngresso,
salario){
297       this->areaSupervisao = areaSupervisao;
298 }
```

virtual Gerente::~Gerente () `[inline], [virtual]`

```
280 {}
```

Funções membros

void Gerente::aumentoSalarial () `[virtual]`

Método aumentoSalarial aumenta o salario do gerente em 10%.

Implementa **Funcionario** (*p.13*).

```
307                                       {
308       this->salario = this->salario + (salario * (10/100));
309       this->tributarSalario();
310 }
```

string Gerente::getAreaSupervisao ()

```
302                                          {
303       return this->areaSupervisao;
304 }
```

void Gerente::setAreaSupervisao (string *AreaSupervisao*)

```
299                                                                    {
300       this->areaSupervisao = areaSupervisao;
301 }
```

A documentação para essa classe foi gerada a partir do seguinte arquivo:

codigo/**main (5).cpp**

Referência da Classe Operador

Diagrama de hierarquia para Operador:

IMAGE

Membros Públicos

Operador ()
Operador (string **nome**, string **endereco**, string **telefone**, string **designacao**, tm **dataIngresso**, double **salario**)
virtual **~Operador** ()
void **aumentoSalarial** ()

Método aumentoSalarial aumenta o salario do operador em 5%.

Outros membros herdados

Construtores e Destrutores

Operador::Operador ()

```
261                          : Funcionario(1){
262
263 }
```

Operador::Operador (string *nome*, string *endereco*, string *telefone*, string *designacao*, tm *dataIngresso*, double *salario*)

```
265      : Funcionario(nome, endereco, telefone, designacao, dataIngresso,
salario){
266
267 }
```

virtual Operador::~Operador () `[inline]`, `[virtual]`

```
256 {}
```

Funções membros

void Operador::aumentoSalarial () `[virtual]`

Método aumentoSalarial aumenta o salario do operador em 5%.

Implementa **Funcionario** (*p.13*).

```
269                                    {
270      this->salario = this->salario + (salario * (5/100));
271      this->tributarSalario();
272 }
```

A documentação para essa classe foi gerada a partir do seguinte arquivo:

codigo/**main (5).cpp**

Referência da Classe Presidente

Diagrama de hierarquia para Presidente:

IMAGE

Membros Públicos

Presidente ()
Presidente (string areaFormacao, string formacaoMaxima, string **nome**, string **endereco**, string **telefone**, string **designacao**, tm **dataIngresso**, double **salario**)
virtual **~Presidente** ()
void **setAreaFormacao** (string AreaFormacao)
void **setFormacaoMaxima** (string formacaoMaxima)
string **getAreaFormacao** ()
string **getFormacaoMaxima** ()
void **aumentoSalarial** ()
Método aumentoSalarial aumenta o salario do presidente em 30%.

Outros membros herdados

Construtores e Destrutores

Presidente::Presidente ()

```
381                                 : Funcionario(4){
382
383 }
```

Presidente::Presidente (string *areaFormacao*, string *formacaoMaxima*, string *nome*, string *endereco*, string *telefone*, string *designacao*, tm *dataIngresso*, double *salario*)

```
385       : Funcionario(nome, endereco, telefone, designacao, dataIngresso,
salario){
386       this->areaFormacao = areaFormacao;
387       this->formacaoMaxima = formacaoMaxima;
388 }
```

virtual Presidente::~Presidente () [inline], [virtual]

```
366 {}
```

Funções membros

void Presidente::aumentoSalarial () [virtual]

Método aumentoSalarial aumenta o salario do presidente em 30%.

Implementa **Funcionario** (*p.13*).

```
403                                           {
404       this->salario = this->salario + (salario*(30/100));
405       this->tributarSalario();
406 }
```

string Presidente::getAreaFormacao ()

```
395                                             {
396       return this->areaFormacao ;
397 }
```

string Presidente::getFormacaoMaxima ()

```
398                                                 {
399       return this->formacaoMaxima;
400 }
```

void Presidente::setAreaFormacao (string *AreaFormacao*)

```
389                                                                 {
390       this->areaFormacao = areaFormacao;
391 }
```

void Presidente::setFormacaoMaxima (string *formacaoMaxima*)

```
392                                                                         {
393       this->formacaoMaxima = formacaoMaxima;
394 }
```

A documentação para essa classe foi gerada a partir do seguinte arquivo:

codigo/**main (5).cpp**

Arquivos

Referência do Arquivo codigo/main (5).cpp

```
#include <iostream>
#include <string>
#include <cstdlib>
#include <vector>
#include <QVector>
#include <exception>
#include <iomanip>
#include <ctime>
```

Componentes

class **Funcionario**
class **Operador**
class **Gerente**
class **Diretor**
class **Presidente**
class **Empresa**

Funções

tm **avancarDia** (tm &data_inicio)
NÃO ESTÃO INCLUSOS ANOS BISSEXTOS.

int **main** ()

Funções

tm avancarDia (tm & *data_inicio*)

NÃO ESTÃO INCLUSOS ANOS BISSEXTOS.

O método avancarDia avanca um dia em relacao a data de inicio dada como parametro e retorna essa data em forma de struct tm disponibilizada na biblioteca ctime>.

```
448                                     {
449       data_inicio.tm_mday++;
450
451       if   ((data_inicio.tm_mon == 2 && data_inicio.tm_mday > 28) ||
452
453            ((data_inicio.tm_mon == 4 || data_inicio.tm_mon == 6 ||
454            data_inicio.tm_mon == 9 || data_inicio.tm_mon == 11)
455            && data_inicio.tm_mday > 30)||
456
457            ((data_inicio.tm_mon == 1 || data_inicio.tm_mon == 3 ||
458            data_inicio.tm_mon == 5 || data_inicio.tm_mon == 7 ||
459            data_inicio.tm_mon == 8 || data_inicio.tm_mon == 10 ||
460            data_inicio.tm_mon == 12) && data_inicio.tm_mday > 31)){
461
462            data_inicio.tm_mday = 1;
463            data_inicio.tm_mon++;
464
465            if (data_inicio.tm_mon > 12){
466                 data_inicio.tm_mon = 1;
467                 data_inicio.tm_year++;
468            }
469       }
470       return data_inicio;
471 }
```

int main ()

```
709             {
710
711 }
```

Sumário

INDEX