Checagem de Tipos

Erros de Tipos em Pascal Simplificado não capturados pela gramática

Tarefas de um “Type Checker” e Conversões de Tipos

Sistema de Tipos

Checagem Estática X Dinâmica

Linguagens Fortemente Tipadas/Tipificadas

Detalhes de Implementação de um “Type Checker”

Erros de Tipo em Pascal Simplificado

- Como a gramática do Pascal Simplificado não faz distinção entre expressões inteiras e booleanas
  - esta distinção deve ser feita pelo compilador de modo a excluir construções como, por exemplo:

```
a[not b] := 2
x := y;      .... sendo x uma inteira e y uma booleana
while (1 > true) and (5) do
while 1 do
if x then  ....sendo x uma variável inteira
p(1,true);     ....tendo sido p definida com o primeiro
                   parâmetro booleano e o segundo
                   parâmetro inteiro
```

Tarefas da Checagem de Tipos (1)

.

- Checagem de tipos é a fase da compilação responsável por:
  - verificar se o tipo de uma expressão “casa” com o esperado em seu contexto.

- Para fazer a Checagem de Tipos um compilador precisa atribuir um parâmetro de tipo a cada componente de um programa fonte que processa expressões
  - então checar se estas expressões de tipos estão de acordo com a coleção de regras lógicas chamada de sistema de tipos da linguagem fonte.

Exemplos

- Operador mod do Pascal requer tipos inteiros
  - Portanto, a checagem de tipos deve verificar se os operandos são inteiros

- Indexação somente pode ser aplicada a um objeto definido como sendo do tipo array e só pode ter um campo se uma variável for do tipo registro:
  - Portanto, haverá ponto de checagem no id da regra quanto ao seu tipo tanto se houve abertura de [quando não devia ou se faltou quando necessário; o mesmo vale para campo

    <variável> ::= <identificador> [**[** <expressão> **]** **|** {**.** <campo>}]

- Os tipos dos argumentos (de procedimento e funções) formais devem ser compatíveis com os reais
  - Portanto, haverá checagem para cada expressão da lista de expressões abaixo:

    <chamada de procedimento> ::= <identificador> [**(** <lista de expressões> **)**]

- OBS: os colchetes em azul acima são da metalinguagem (significando opcionalidade)

Tarefas da Checagem de Tipos (2)

- A checagem de tipos é também importante como recurso auxiliar para as atividades de geração de código

- Na maioria das linguagens a utilização simultânea de dados cujos tipos são diferentes embora coerentes dá ao gerador condições para
  - geração de um código contendo as conversões eventualmente necessárias para permitir que sejam realizadas adequadamente as operações.

Conversões

- Implícitas (coersões): feitas automaticamente pelo compilador, geralmente quando não se perde informação (Real:= Integer; String:= Char; Char := String[1])

  Por isso, i:= r dará ERRO

- Explícitas: o programador usa funções pré-definidas para realizar as conversões (trunc, int, ord, char, ...)

- A definição da linguagem especifica quais as conversões necessárias .
  - Em expressões a conversão usual é de inteiro para real e daí executa-se a operação real sobre os operandos

    "Os operadores reais são usados para formar expressões que contém somente operandos do tipo REAL ou também misturados com tipos INTEGER. O resultado será sempre REAL."

- O checador de tipos pode inserir essas operações de conversões no código intermediário, gerando um código objeto já convertido

- Se as conversões implícitas forem feitas pelo compilador o código objeto será muito mais otimizado
  - Suponha X real:
    - For i := 1 to n do X[I] := 1 (48.4 microseg)
      - A conversão é feita em tempo de execução MAS poderia já ser feita em tempo de compilação desde que X é real
    - For i := 1 to N do X[i] := 1.0 (5.4 microseg)

Operadores Sobrecarregados

- Conversões de tipos são também importantes quando temos operadores e funções sobrecarregados

- Os operadores aritméticos, por exemplo +, são sobrecarregados em muitas linguagens e resolvidos olhando-se seus argumentos para se decidir que versão de + deve-se usar.

E → E1 op E2

```
E.type := if E1.type = int and E2.type = int
                then int
                else if E1.type = int and E2.type = real
                        then real
                        else if E1.type = real and E2.type = int
                                then real
                                else if E1.type = real and E2.type = real
                                        then real
                                        else if E1.type = char and E2.type = char
                                                then char else erro
```

OBS: + também significa concatenação de 2 strings ou chars.

Sistema de Tipos

- O projeto de um “type checker” está fundamentado em:
  - informações sobre os construtores sintáticos da linguagem (descritores para array, record, ponteiros, etc. já vistos),
  - a noção de tipos, e
  - as regras para se atribuir tipos aos construtores, chamadas de Sistema de Tipos.
- Dentro dele vemos também as regras de equivalência de expressões.

Sistema de Tipos

Um “type checker” implementa um sistema de tipos.

- Exemplo de Regra do Relatório Pascal que ajuda a começar a implementação:
  - “Se ambos operandos dos operadores aritméticos de adição, subtração e multiplicação são do tipo inteiro então o resultado será inteiro.”

Tipo de uma Variável

- Dois conceitos relacionados são importantes:
  - declaração implícita/explícita de tipo e
  - amarração estática/dinâmica de tipo a variáveis.

Tipo de declaração e formas de amarração

- Em Pascal/C/Fortran a amarração entre uma variável e seu tipo é especificada por uma declaração explícita de variável
  - Fortran permite também a declaração implícita dada pela letra de início do identificador
  - A declaração explícita garante maior confiabilidade aos programas.

- APL (L. funcional) permite amarração dinâmica entre variáveis e tipos.
  - O nome de uma variável pode, em pontos diferentes, assumir vários tipos, dependendo do valor que ela assume.
  - As variáveis não são declaradas implicitamente, é SEU TIPO que é determinado implicitamente em função de seu valor corrente. Isto requer verificação dinâmica de tipo

- Amarração estática é a base para a verificação estática de tipos
  - os descritores existem só em tempo de compilação.

  - Linguagens com amarração dinâmica são mais adaptadas à Interpretação
  - Linguagens com amarração estática são mais adaptadas à Compilação

- A checagem de tipos feita pelo compilador é chamada estática
  - enquanto que a feita em tempo de execução é dinâmica

(Aho, Lam, Sethi, Ulman, 07)

Uma linguagem é fortemente tipada se o compilador garante que os programas aceitos por ele irão executar sem erros de tipos.

(Sebesta, 2002)

- Uma linguagem de programação é dita "fortemente tipificada" se seus erros de tipos sempre forem detectados

- Os tipos de todos os operandos podem ser determinados, seja em tempo de compilação ou em tempo de execução

Tipificação Forte

- FORTRAN 77 não é fortemente tipificada
  - parâmetros formais e reais
  - EQUIVALENCE
- Pascal quase fortemente tipificada
  - registros variantes não são verificados, p.e.
- C e C++ não são:
  - verificação de tipos de parâmetros pode ser evitada
  - "unions" não são verificados

- Ada é praticamente fortemente tipificada
  - registros variantes são verificados
  - função UNCHECKED_CONVERSION permite suspensão temporária da verificação de tipos
- ML é fortemente tipificada
  - identificadores são amarrados a tipos estaticamente (declaração), ou
  - tipos são reconhecidos via inferência
- Java é fortemente tipificada como Ada

Pascal não é fortemente tipada????

Não, pois:

1) Em Pascal, intervalos não podem ser testados estaticamente (e nem em qualquer outra linguagem).

- Em a := b + c, sendo as variáveis declaradas no intervalo de 1..10, não se pode garantir a priori que o valor de b+c pertence ao intervalo de 1..10

- Em V[I] := ..., sendo V: array [1..255] of ..., não se garante que durante a execução o valor de i irá permanecer no intervalo de 1..255.
  - Como o tipo do índice de um vetor é parte do tipo do vetor, o acesso ao vetor requer um teste em tempo de execução para verificar se o índice está dentro dos limites.

Uma linguagem fortemente tipada

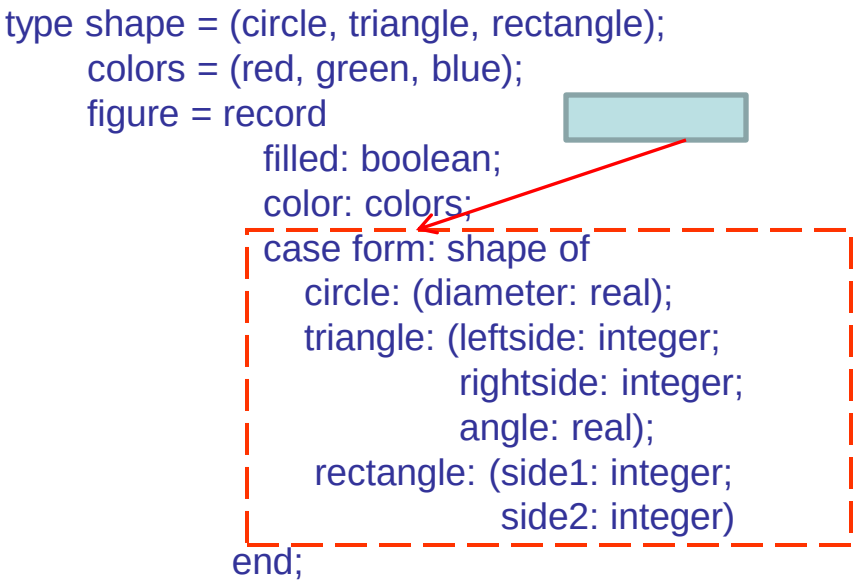
- Algol 68 não considera os limites como parte do TIPO e sim como parte do VALOR. Assim, os índices são ignorados em tempo de compilação, sendo inseridos testes para avaliação em tempo de execução.

2) Não é possível testar estaticamente o emprego correto dos registros com variantes de Pascal e testes em tempo de execução raramente são feitos por causa do impacto na eficiência.

Pascal permite que o componente discriminante seja eliminado

– Em contraste, a união em ADA é totalmente segura

Exige que o discriminante receba atribuição quando o componente da união receber



Record Variante sem o discriminante (escapes)

It may be possible to do this in Pascal using an undiscriminated variant record:

```
var a: integer;
    b: real;
    a2c: record
           case boolean of
             false: (a: integer);
             true:  (b: real);
           end;
         end;
begin
  a2c.b := b;
  a := a2c.a;
end;
```

Although casting is possible on the most of Pascal compilers and interpreters, even in the code above a2c.a and a2c.b aren't required by any Pascal standardizations to share the same address space. Niklaus Wirth, the designer of Pascal, has written about the problematic nature of attempting type escapes using this approach:

"Most implementors of Pascal decided that this checking would be too expensive, enlarging code and deteriorating program efficiency. As a consequence, the variant record became a favourite feature to breach the type system by all programmers in love with tricks, which usually turn into pitfalls and calamities".

Several languages now specifically exclude such type escapes, for example Java, C# and Wirth's own Oberon.

3) Não há regras de compatibilidade de tipos rigorosamente especificadas no Relatório Pascal.

- Em contraste, Algol 68 se esforçou para definir a noção de compatibilidade precisamente.
  - Assim, as checagens em Pascal variam de implementação para implementação.

Regras de Compatibilidade/Equivalência

- As regras de checagem de tipos são geralmente da forma:

  "Se duas expressões são equivalentes então retorne um certo tipo senão erro"

- Precisamos então ter uma definição precisa de quando duas expressões são equivalentes

- Encontramos duas noções de compatibilidade nas linguagens:
  - equivalência de nomes e
  - equivalência estrutural

- Nomes: duas variáveis possuem tipos compatíveis se:
  - têm o mesmo nome do tipo, definido pelo usuário ou primitivo,

  ou
  - aparecem na mesma declaração.

- Estrutural: duas variáveis tem tipos compatíveis se possuem a mesma estrutura.
  - Os tipos definidos pelo usuário são usados só como abreviatura da estrutura que representam e não introduzem qualquer característica semântica nova.

  - Para se checar a equivalência, os nomes dos tipos definidos pelo usuário são substituídos pelas suas definições repetidamente até não sobrarem mais tipos definidos pelo usuário. (recursão)

Exemplos

```
Type t = array [1..20] of integer;
Var a,b : array [1..20] of integer;
      c: array [1..20] of integer;
      d: t;
       e,f: record
              a: integer;
              b: t
            end;
```

```
Type t = array [1..20] of integer;
Var a,b : array [1..20] of integer;
      c: array [1..20] of integer;
      d: t;
       e,f: record
              a: integer;
              b: t
            end;
```

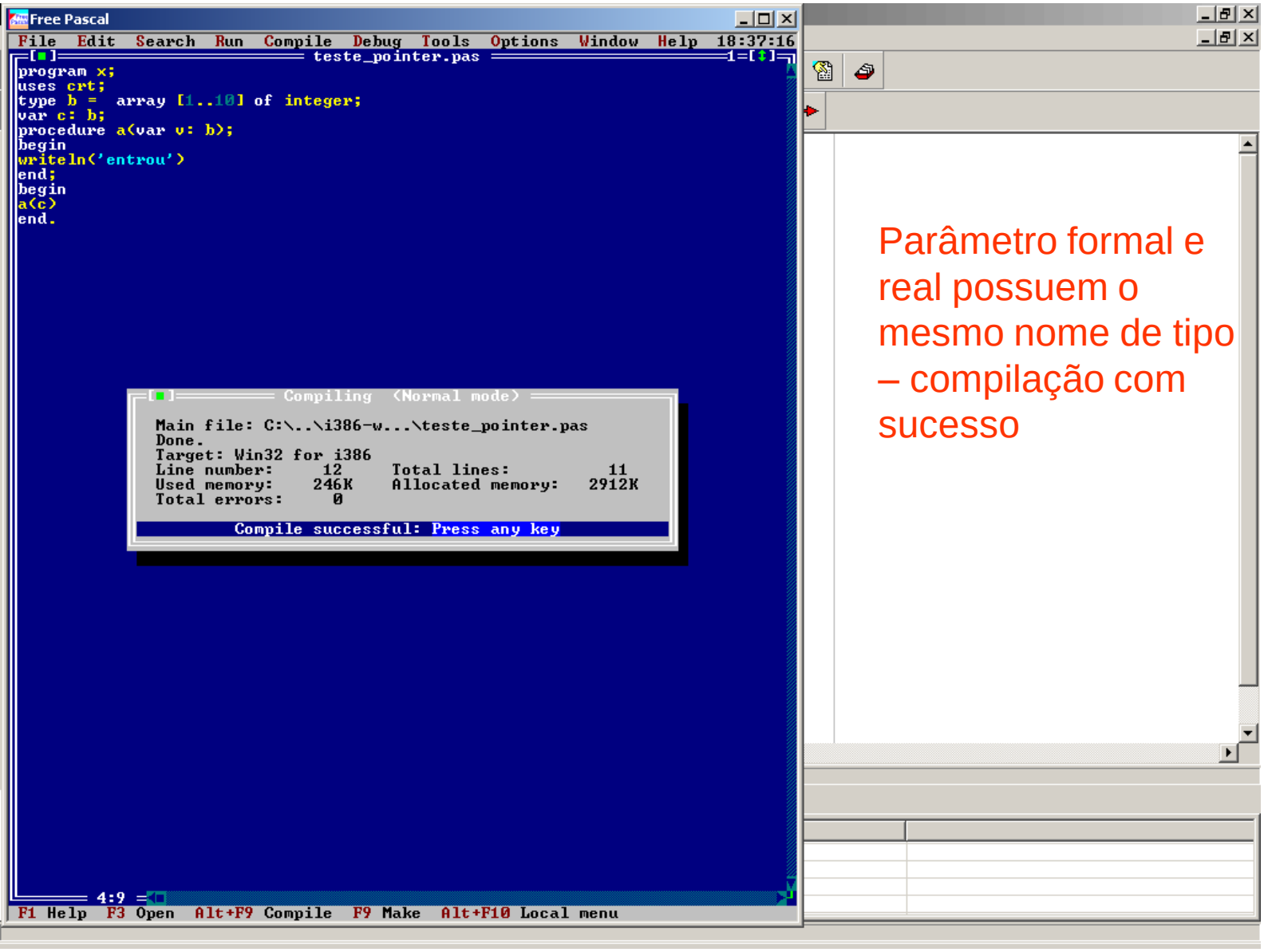
- (a e b); (e e f); (d, e.b e e.f) tem equivalência de nome
- a, b, c, d, e.b, f.b tem tipos compatíveis estruturalmente

Vejam a variação de resultados

- No Relatório de Pascal os parâmetros e o retorno das funções
  - possuem equivalência de nomes (exigem um nome de tipo) e
  - o resto é estrutural.
- porém, para 2 compiladores Pascal
  - Free-Pascal
  - Dev-Pascal
- os assuntos acima são tratados de forma diferente.
  - Dev-Pascal se adere mais ao Relatório Pascal

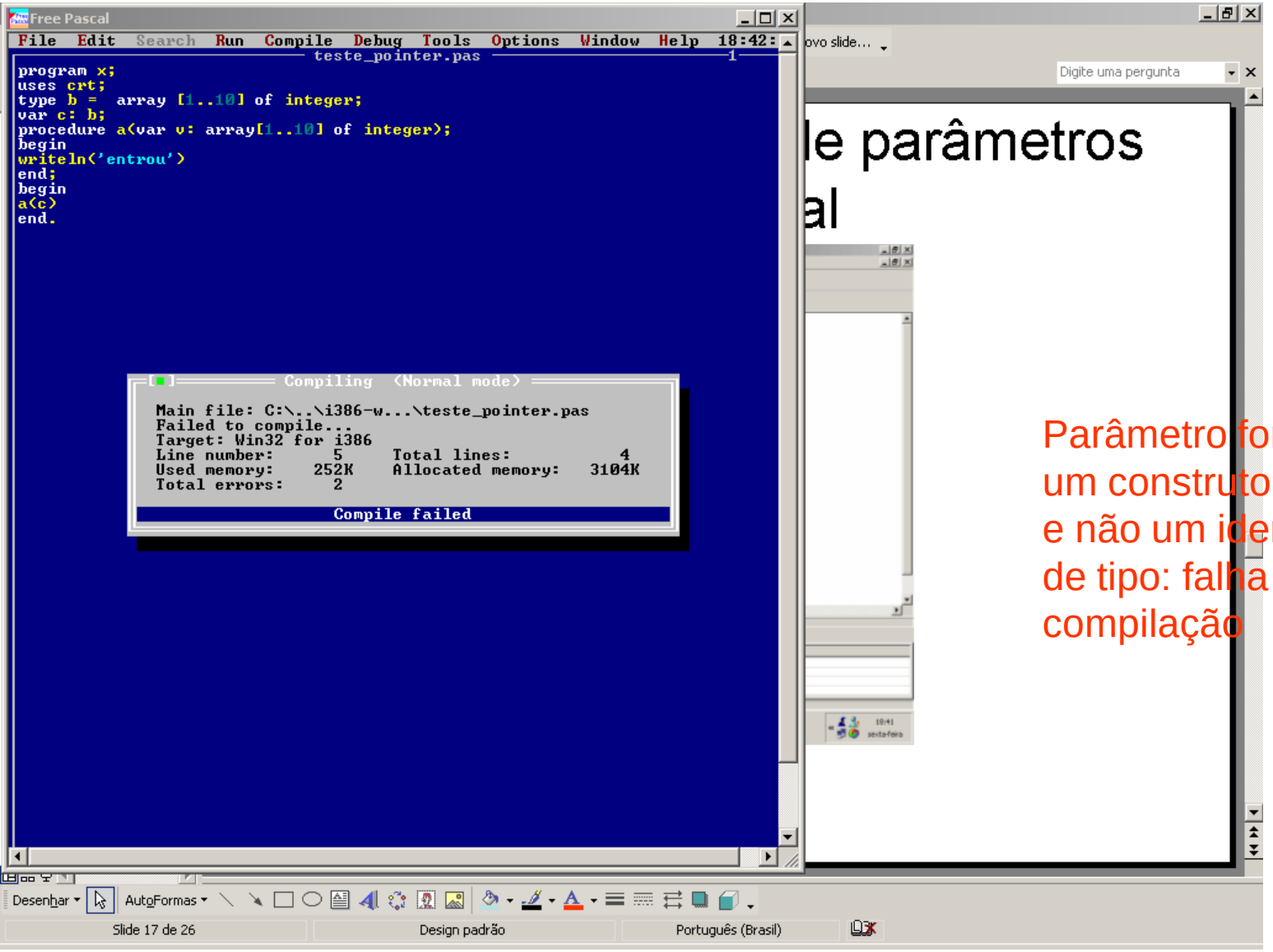
Free-Pascal

Checagem de Parâmetros:Free-Pascal



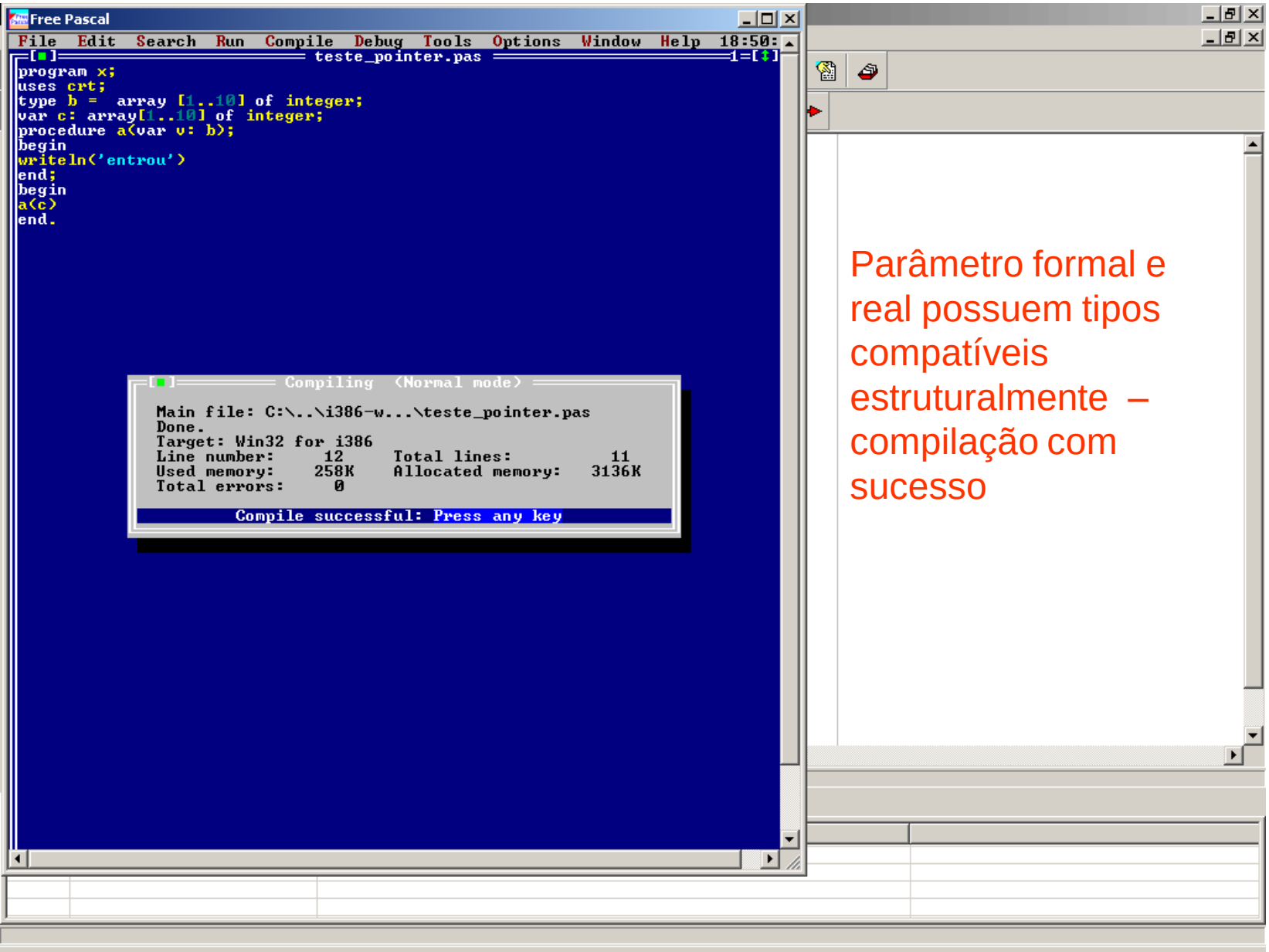
Parâmetro formal e real possuem o mesmo nome de tipo – compilação com sucesso

Checagem de Parâmetros:Free-Pascal



Parâmetro formal usa um construtor de tipo e não um identificador de tipo: falha na compilação

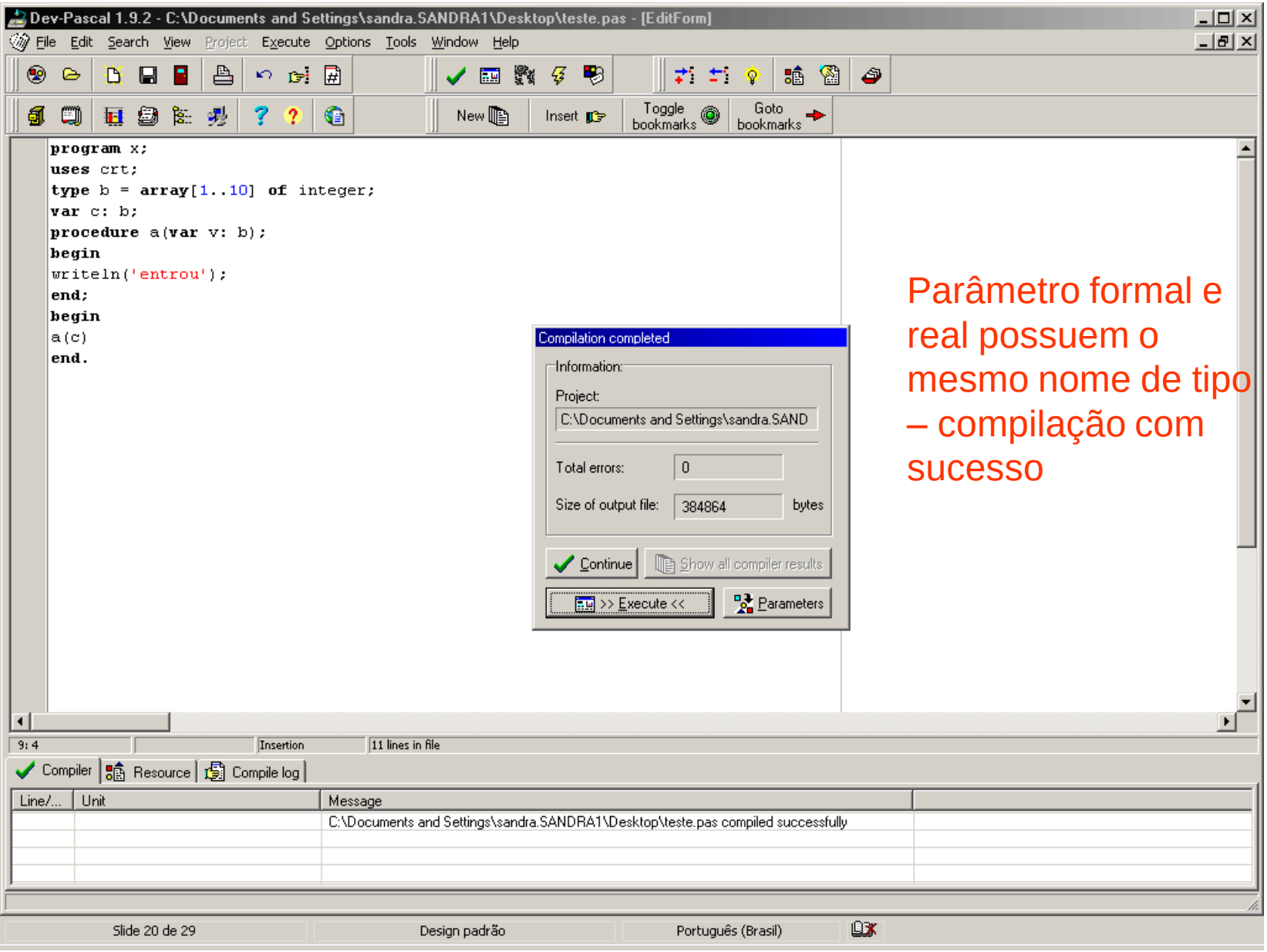
Checagem de Parâmetros:Free-Pascal



Parâmetro formal e real possuem tipos compatíveis estruturalmente – compilação com sucesso

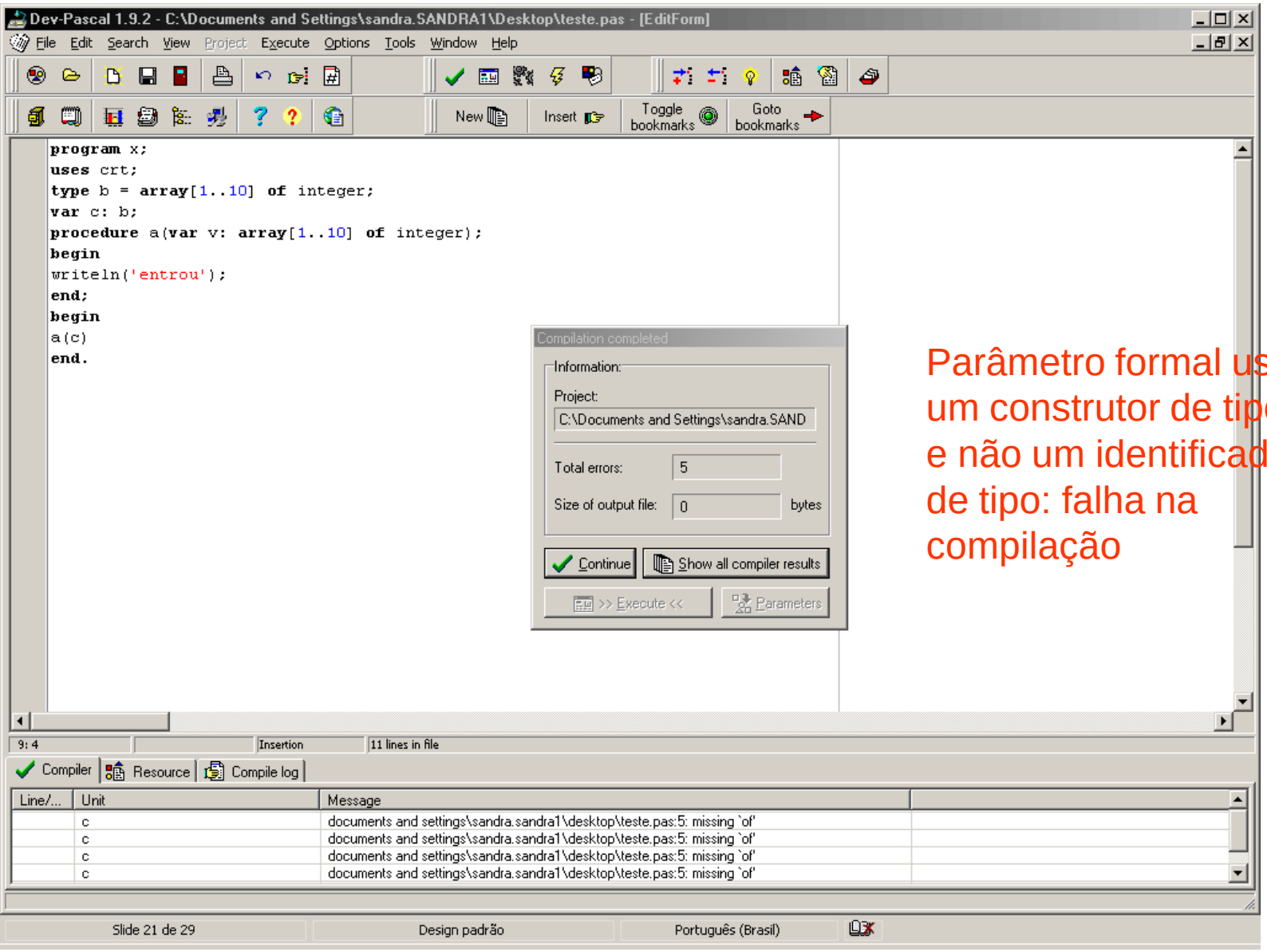
DEV-PASCAL

Checagem de Parâmetros: Dev-Pascal



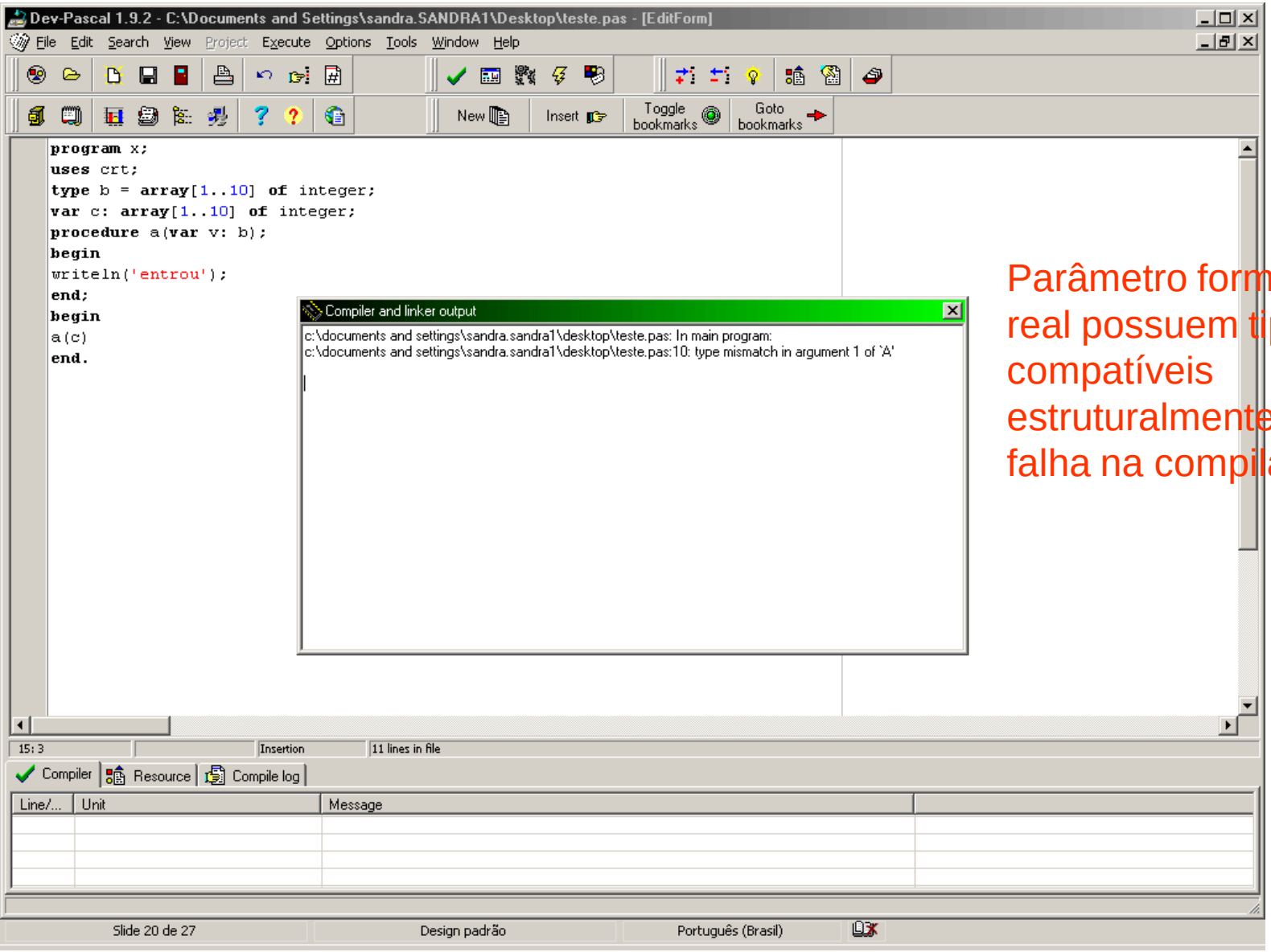
Parâmetro formal e real possuem o mesmo nome de tipo – compilação com sucesso

Checagem de Parâmetros: Dev-Pascal



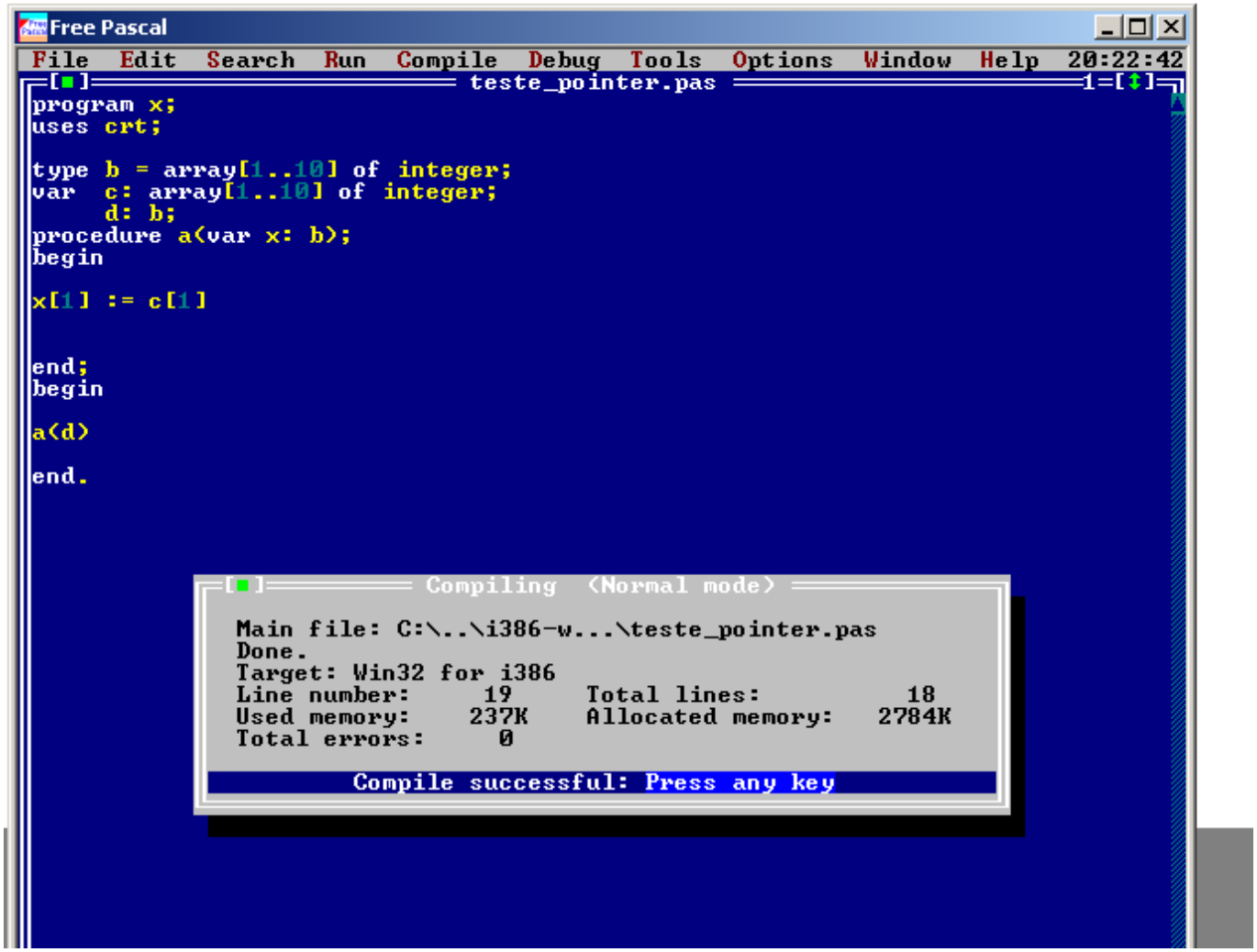
Parâmetro formal usa um construtor de tipo e não um identificador de tipo: falha na compilação

Checagem de Parâmetros: Dev-Pascal



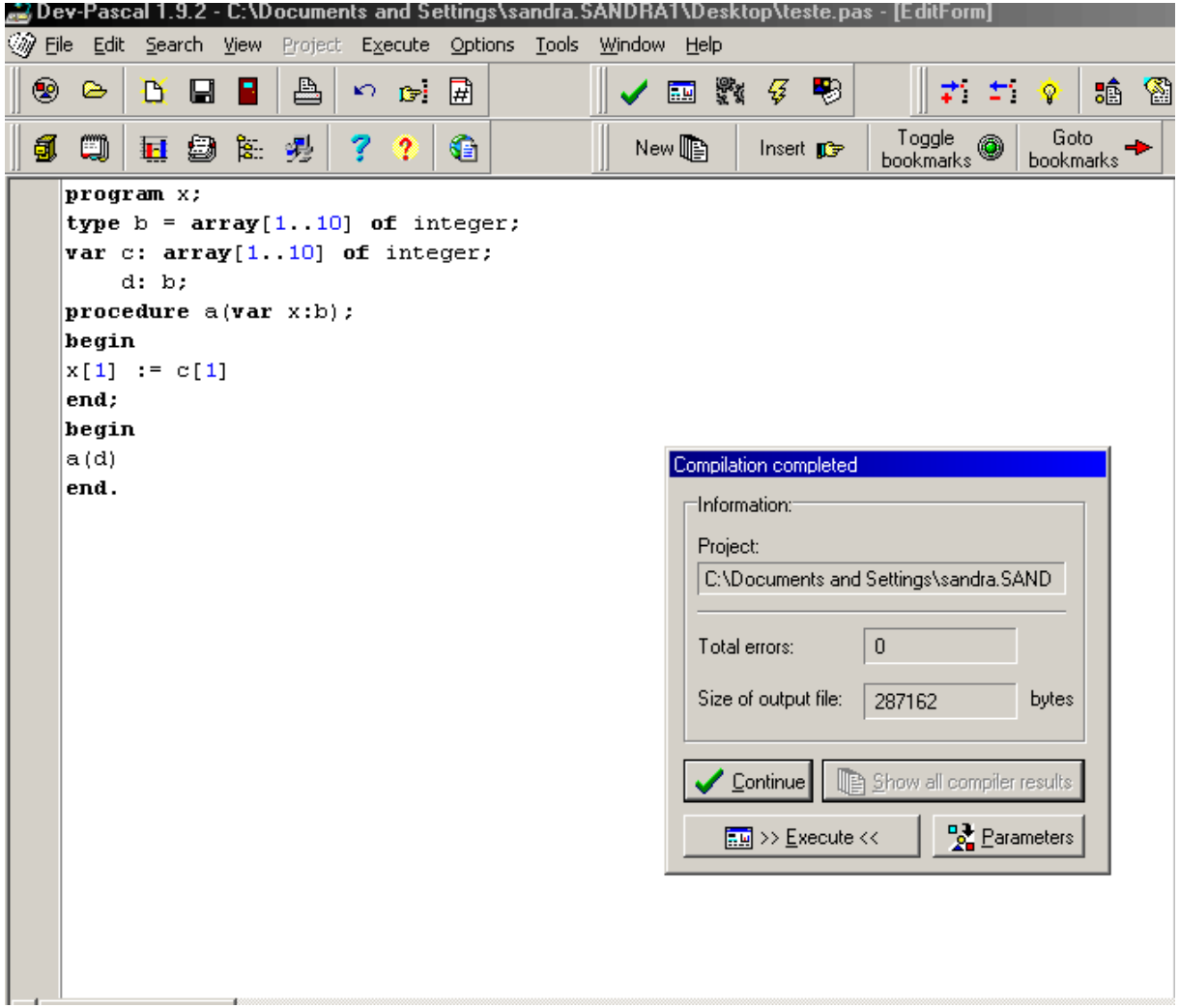
Parâmetro formal e real possuem tipos compatíveis estruturalmente – falha na compilação

A checagem do uso do parâmetro: Free-Pascal



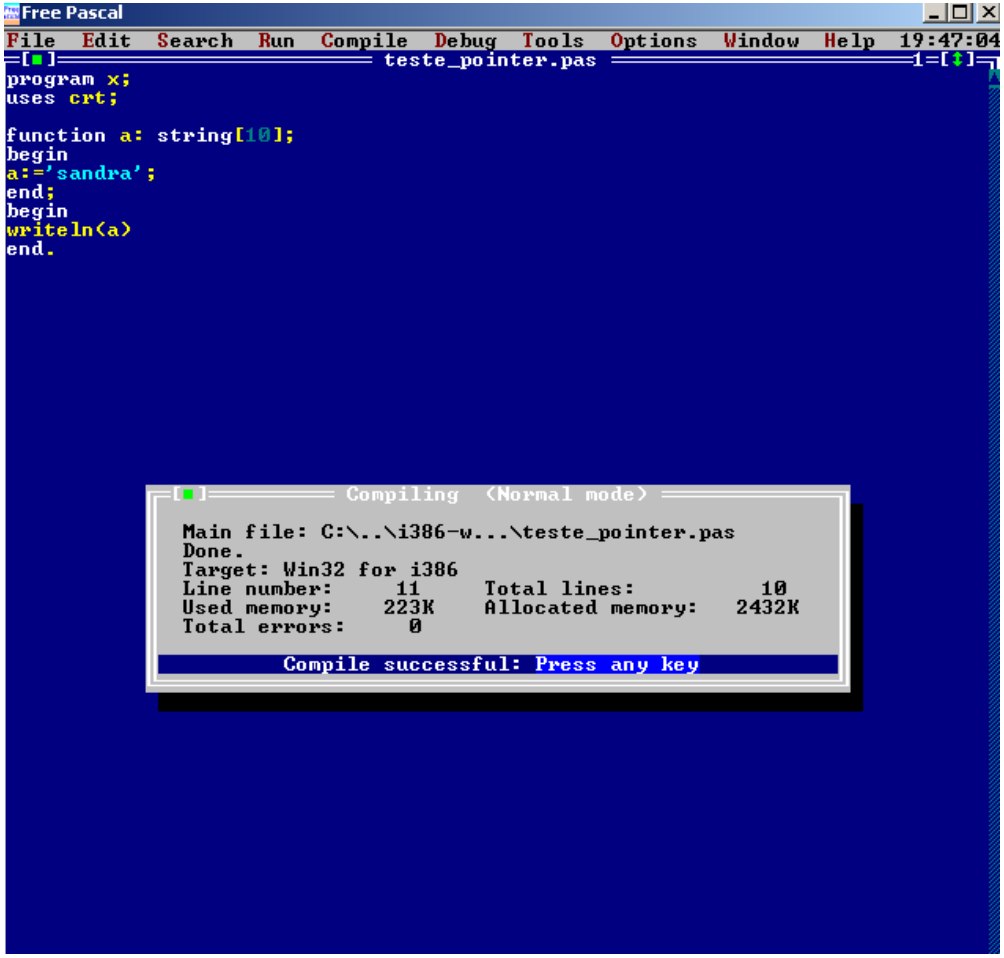
Comandos de atribuição com os parâmetros possuem checagem estrutural

A checagem do uso do parâmetro: Dev-Pascal



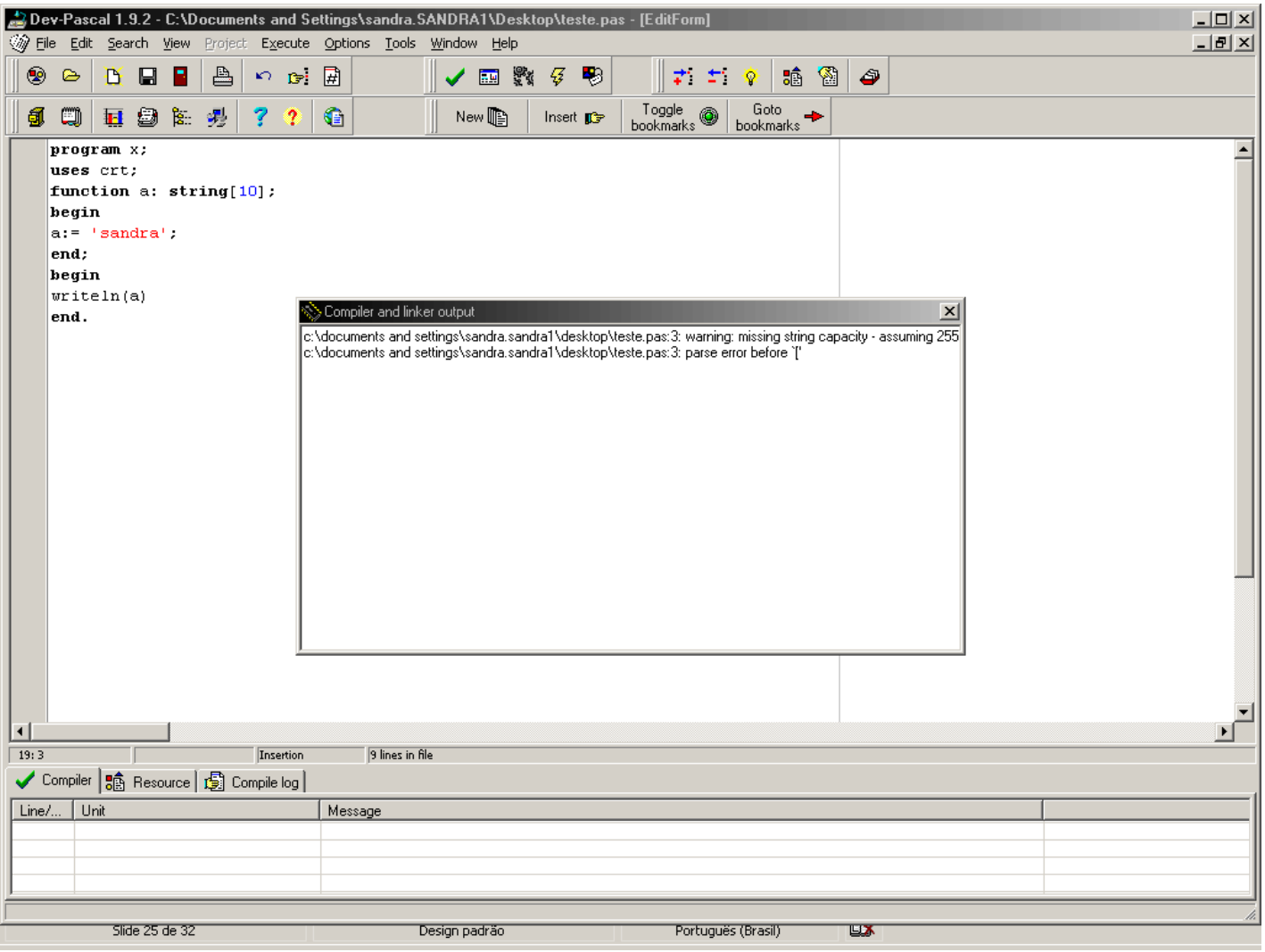
Comandos de atribuição com os parâmetros possuem checagem estrutural

Checagem de retorno de função: Free-Pascal



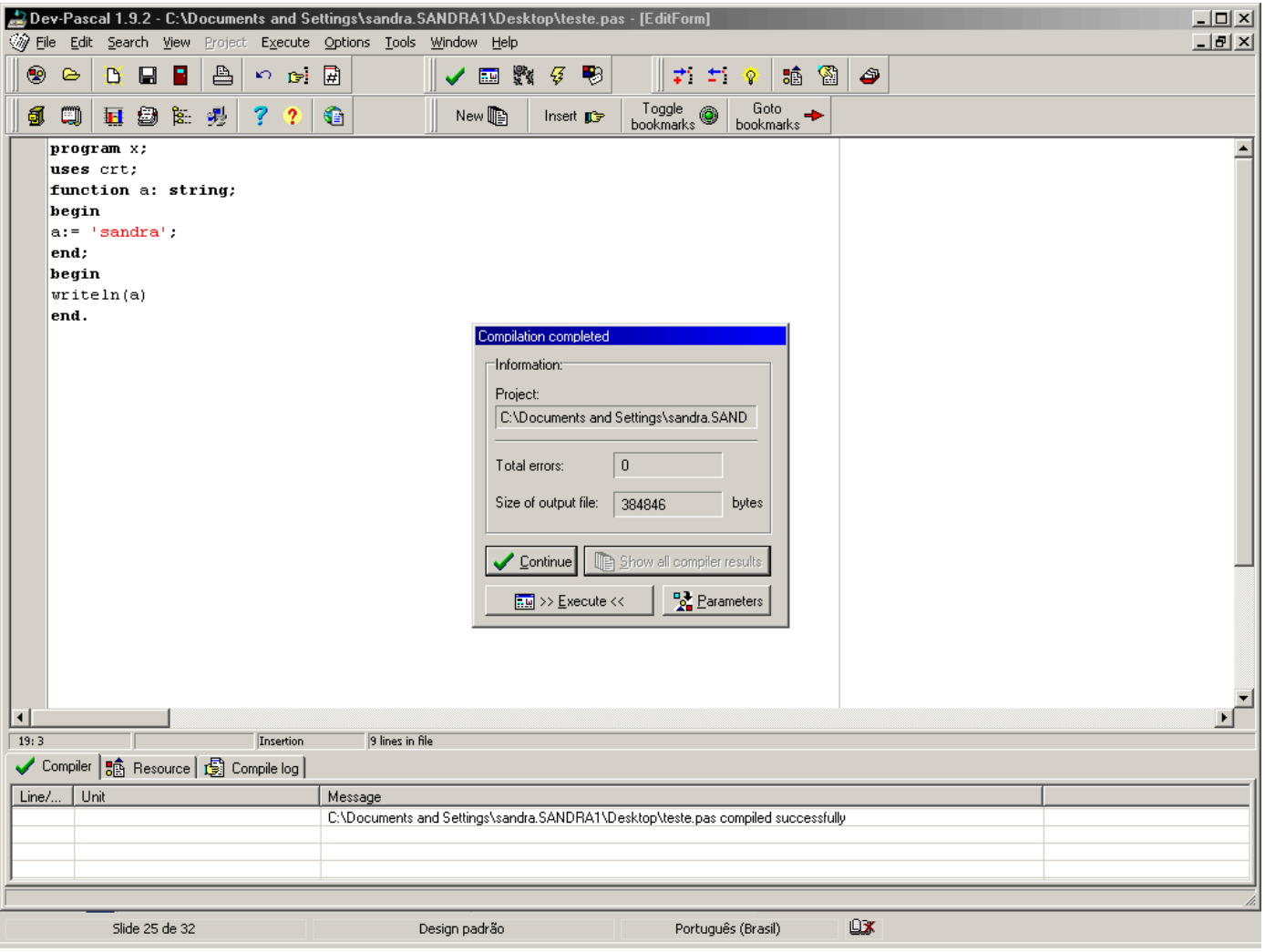
Permite string com tamanho limitado como tipo de função

Checagem de retorno de função: Dev-Pascal



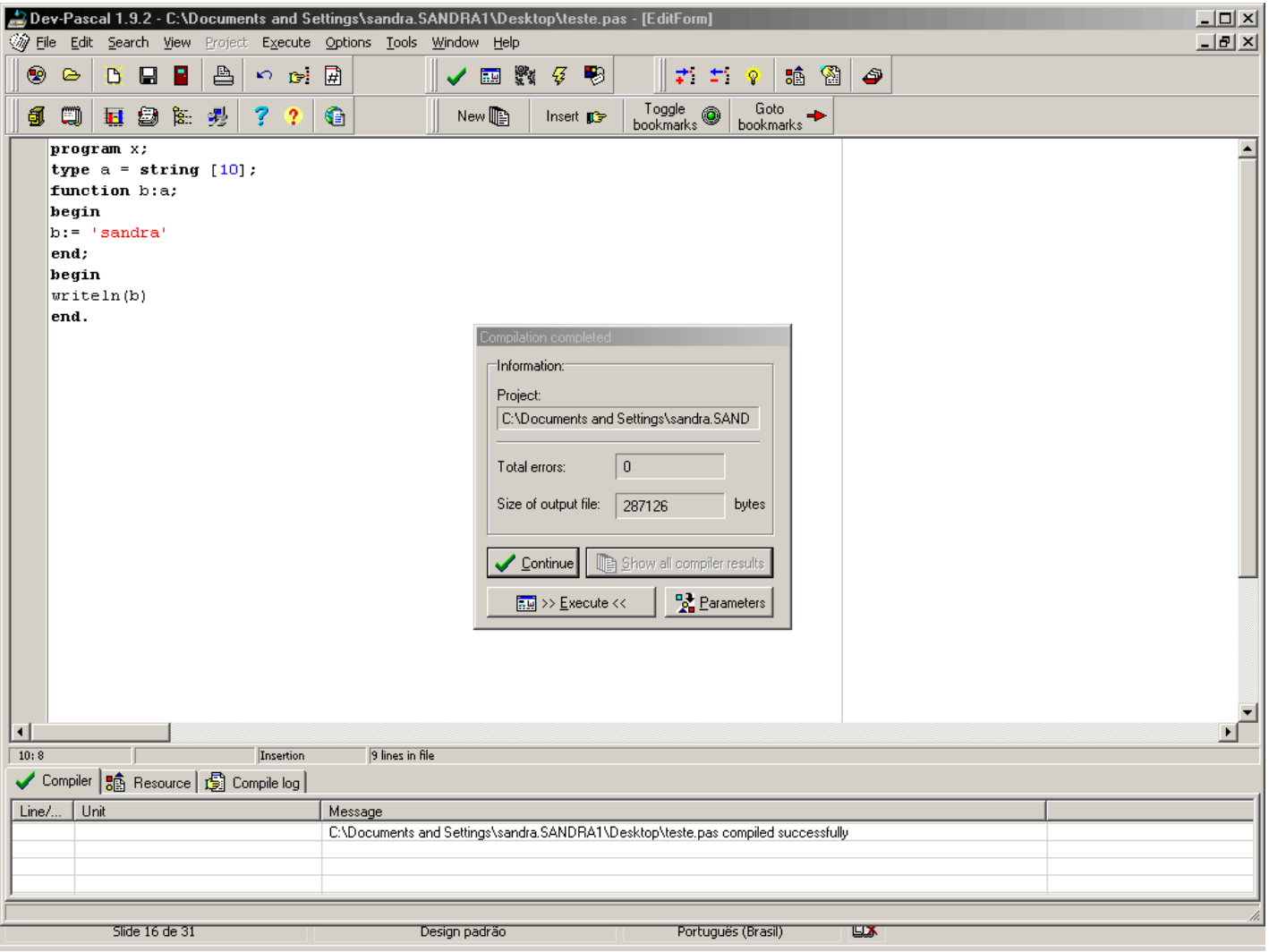
Não permite string com tamanho limitado como tipo de função

Checagem de retorno de função: Dev-Pascal



Permite string sem tamanho limitado como tipo de função

Checagem de retorno de função: Dev-Pascal



Tipo de função com um nome de tipo: sucesso na compilação

Implementação da Checagem de Tipos

- Cap. 10 do Kowaltowsky
  - as rotinas: termo, fator, atribuição, comando condicional (IF) implementadas para o PS (tipo inteiro e booleano)

Exercício

```
procedure termo ;

begin
  Fator;
  Enquanto simbolo in [*,/, and] faça
  begin
    simbolo := analex(s);
    Fator;
  end
end;
```

Rotina Termo

```
procedure termo (var t: string);
var t1, t2: string;
begin
  Fator (t);
  Enquanto simbolo in [*,/, and] faça
  begin
    s1:= simbolo;
    simbolo := analex(s);
    Fator(t1);
    Caso s1 seja
        * : t2 := ´inteiro´;
        /: t2 := ´inteiro´;
        and : t2 := ´booleano´
    end;
    Se (t <> t1) ou (t <> t2) então erro(´incompatibilidade de tipos´)
  end
end;
```

Rotina FATOR

```
procedure fator (var t: string);
Inicio
Caso simbolo seja
Número: {t:= ´inteiro´; simbolo := analex(s);}
Identificador: {Busca(Tab: TS; id: string; ref: Pont_entrada; declarado: boolean);
                Se declarado = false então erro;
                Obtem_atributos(ref: Pont_entrada; AT: atributos);
                Caso AT.categoria seja
                   Variavel: {t:= AT.tipo; simbolo := analex(s);}
                   parametro: {t:= AT.tipo; simbolo := analex(s);}
                   constante: {t:= 'boolean'; simbolo := analex(s);}
                Else erro;
                fim caso;
Cod_abre_par: {simbolo := analex(s); expressao(t); se simbolo <> Cod_fecha_par
    then erro; simbolo := analex(s);}
Cod_neg: {simbolo := analex(s); fator(t); se t <> 'booleano' então erro
Else erro;
Fim caso;
```

Neste PS não há declaração de constantes, por isso a categoria constante no case devolve boolean, sem checar na TS

Implementação

- Façam as rotinas expressão simples e expressão de forma equivalente a termo
- Desta forma expressão retornará um parâmetro (T) que é o tipo da expressão.
- Este parâmetro deve ser checado
  - nos comandos if, while, repeat: t deve ser booleano;
  - em atribuição o lado esquerdo deve ser compatível (estruturalmente) com o direito;
  - os parâmetros dos procedimentos também devem ser checados, pois o tipo do parâmetro real deve ser compatível com o tipo do parâmetro formal.
  - Na regra que define <variável> não se esqueçam de avaliar se é simples, de record ou indexada (tipos primitivos ou record ou array)

Comando For

<comando for> ::= **for** <variável de controle> **:=** <valor inicial>
(**to|downto**) <valor final> **do** <comando>

<variável de controle> ::= <identificador>

<valor inicial> ::= <expressão>

<valor final> ::= <expressão>

- **Observações úteis:**
- A <variável de controle> deve ser do tipo ordinal (inteiro, char ou booleano)
- O <valor inicial> e o <valor final> devem ser de um tipo compatível com o da <variável de controle>