Construção de Compiladores em OCaml

José Romildo Malaquias

BCC328: Construção de Compiladores I

Universidade Federal de Ouro Preto
Departamento de Computação

2014–2

SUMÁRIO

Parte I

OCaml

1 Introdução a OCaml

Resumo

OCaml é uma linguagem de programação de uso geral, com ênfase na expressividade e segurança. Desenvolvida há mais de 20 anos no INRIA[1] por um grupo de pesquisadores líderes, OCaml beneficia-se de um dos sistemas de tipos mais avançados e suporta os estilos de programação funcional, imperativo e orientado a objetos, que facilitam o desenvolvimento de um software flexível e confiável. Usado em ambientes onde um único erro pode custar milhões e onde velocidade é importante, OCaml é apoiada por uma comunidade ativa que desenvolveu um rico conjunto de bibliotecas e irá ajudá-lo a tirar o máximo de possibilidades de OCaml.

OCaml é uma linguagem de programação moderna, de alto nível, com muitos recursos úteis.

Neste capítulo você se familiarizará com a linguagem OCaml, que será utilizada no desenvolvimento de um compilador.

Sumário

1 Algumas características de OCaml

A linguagem OCaml oferece:

- um sistema de tipos poderoso
- tipos algébricos definidos pelo usuário e casamento de padrão
- gerenciamento automático de memória
- compilação separada de aplicações autônomas
- um sistema de módulos sofisticado

Visite o site de OCaml no endereço `http://ocaml.org/`. Lá você encontrará várias informações interessantes sobre a linguagem.

2 Instalação das ferramentas de desenvolvimento em OCaml

Para o desenvolvimento de aplicações em OCaml é necessário o compilador de OCaml e um conjunto de bibliotecas. A distribuição de OCaml oferece dois compiladores:

- `ocamlc`: um compilador para bytecode
- `ocamlopt`: um compilador nativo

Existe também um ambiente interativo (`ocaml`) onde o programador pode carregar módulos e avaliar expressões, facilitando o desenvolvimento incremental.

Para preparar os programas é necessário um editor de texto e um terminal, ou um ambiente de desenvolvimento integrado (IDE).

[1] Institut National de Recherche en Informatique et en Automatique

O desenvolvimento baseado na linha de comando usa um terminal e um editor de textos. O programador digita os arquivos fontes em um editor de texto e compila a aplicação em um terminal executando um dos compiladores.

Emacs (juntamente com o plugin Tuareg) é um bom editor de texto com facilidades para desenvolvimento em OCaml. Porém o grau de dificuldade de sua aprendizagem é considerado elevado. Emacs está disponível para a maioria das platformas, incluindo Linux e Windows.

Notepad++ é uma opção mais simples disponível no Windows. No Linux temos outras opções como o gedit (parte do ambiente GNOME) e o kate (parte do ambiente KDE).

Eclipse é um ambiente de desenvolvimento integrado muito usado com várias linguagens de programação como Java e C++. O plugin OcaIDE oferece suporte para desenvolvimento em OCaml no Eclipse.

O compilador de OCaml e suas bibliotecas podem ser instalados de várias maneiras. As mais usadas são:

- usando `OPAM`, um gerenciador de pacotes específicos para OCaml (ainda não disponível para Windows),
- usando um gerenciador de pacotes suportado para sua plataforma, ou
- a partir do código fonte.

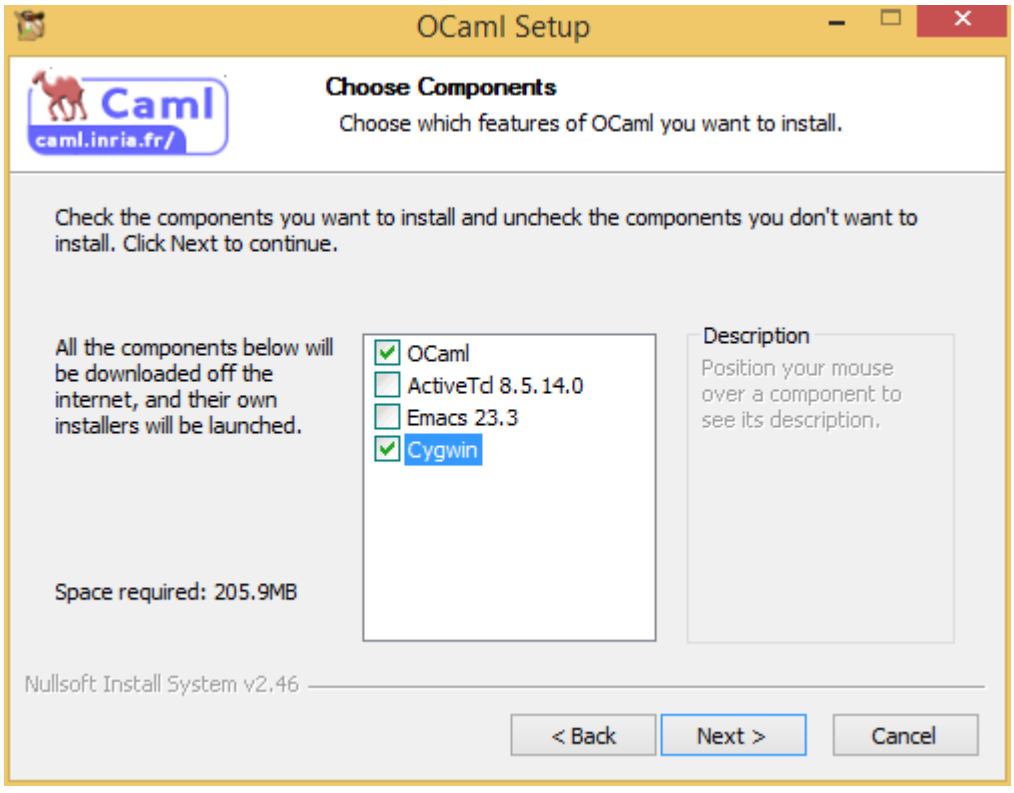
A página Install OCaml apresenta maiores detalhes.

2.1 Instalação do OCaml no Windows

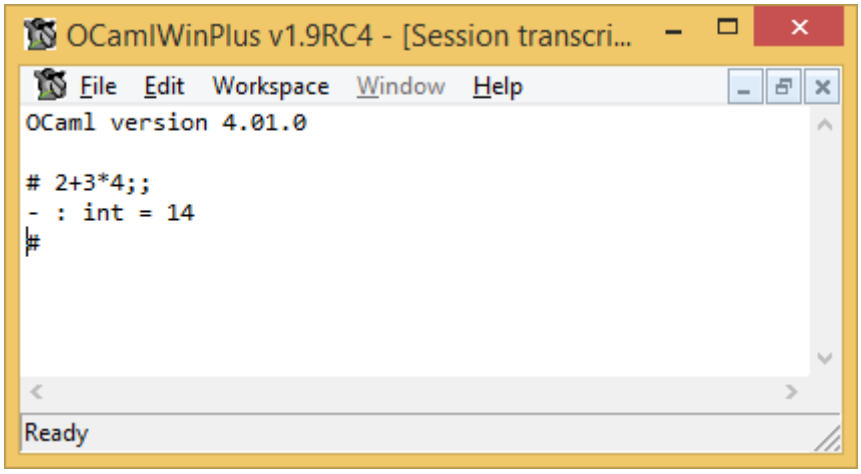
Há várias maneiras de instalar OCaml no Windows. Vamos usar o instalador oficial, disponível na página `http://protz.github.com/ocaml-installer/`.

1. Faça o donwload da versão mais recente do programa de instalação disponível na página `http://protz.github.io/ocaml-installer/`. No momento em que estas instruções foram escritas a versão mais recente era para `OCaml-4.01.0`, acessível pelo link `http://gallium.inria.fr/~protzenk/caml-installer/ocaml-4.01.0-i686-mingw64-installer3.exe`.
2. Execute o programa de instalação, que poderá instalar:
   - OCaml, findlib e flexdll.
   - Emacs (opcional), um editor de texto, com suporte para OCaml.
   - ActiveTCL (opcional), uma biblioteca para aplicações com interface gráfica. Necessário para usar OCamlBrowser ou para criar aplicações com interface gráfica usando Tcl/Tk (lablltk).
   - Cygwin (opcional), uma coleção de ferramentas que permitem que o Windows possa de certa forma agir como um sistema Unix. Necessário para compilação nativa de programas em OCaml. O programa de instalação do Cygwin será executado com os pacotes necessários previamente selecionados.

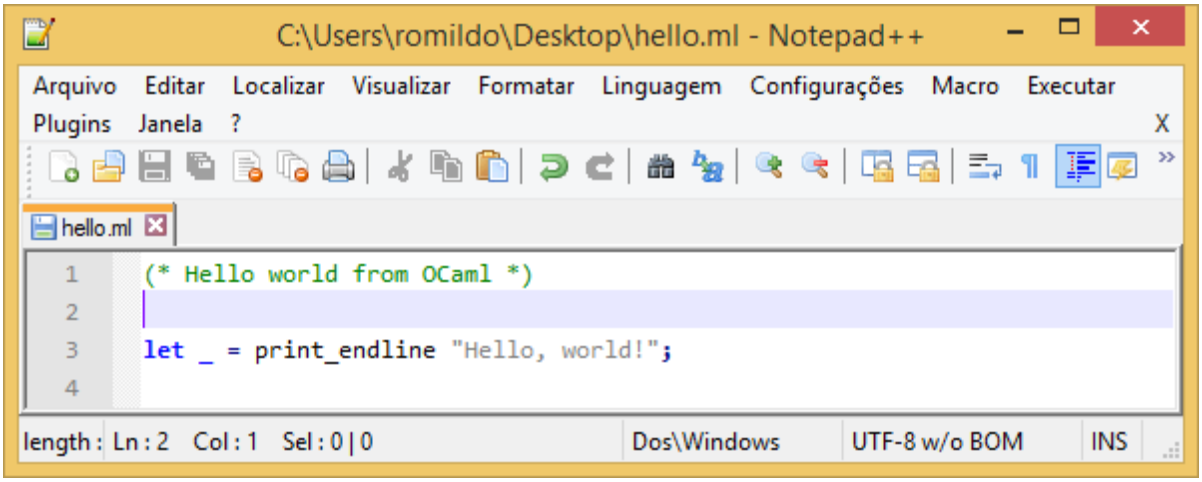
   Certifique-se de habilitar a instalação do Cygwin.



O ambiente interativo poderá ser executado usando o item OCamlWin no menu de programas. Será aberta uma janela onde você pode avaliar expressões e declarações.

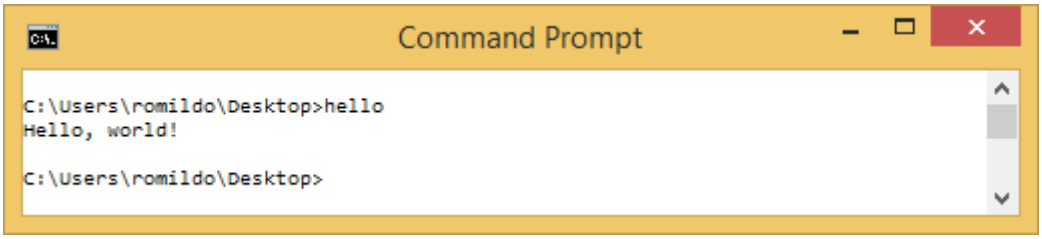


Para desenvolver um programa, edite o texto do programa em um editor de texto, como o Notepad++:



Em seguida compile o programa no terminal:

E execute o programa no terminal:



2.2 Instalação do OCaml no Ubuntu

OCaml é muito fácil de ser instalado no Ubuntu. Use o gerenciador de pacotes para instalar os seguintes pacotes:

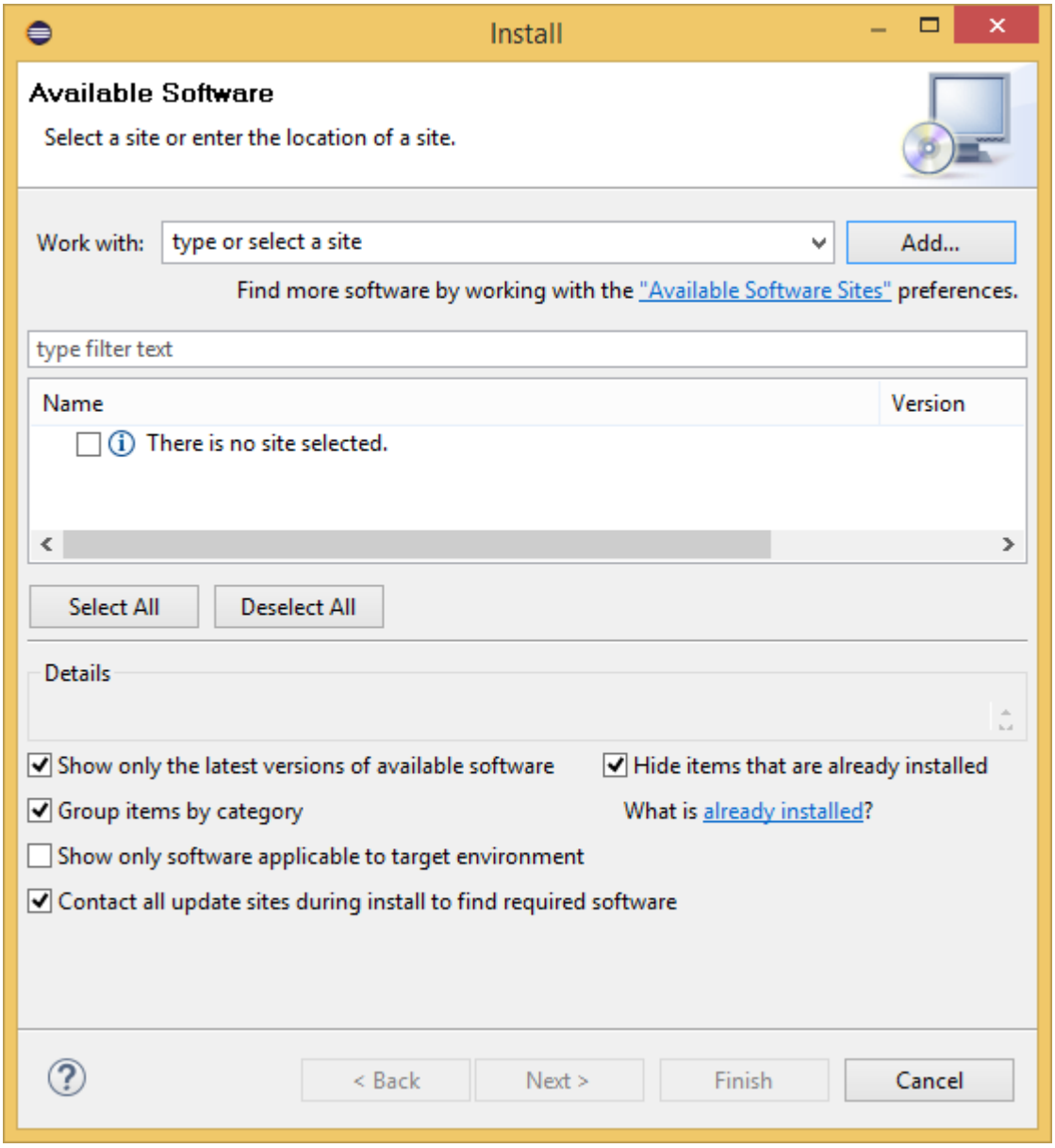
- `ocaml`, para desenvolvimento de aplicações
- `ocaml-native-compilers`, para compilação de programas para código nativo

- `ocaml-doc`, para ter acesso ao manual de referência
- `tuareg-mode`, um plugin para o editor Emacs
- `ocaml-findlib`, para instalar e usar bibliotecas e suas dependências facilmente
- `menhir`, um gerador de analisadores sintáticos
- `rlwrap`, para facilitar a edição da linha de entrada do ambiente interativo,

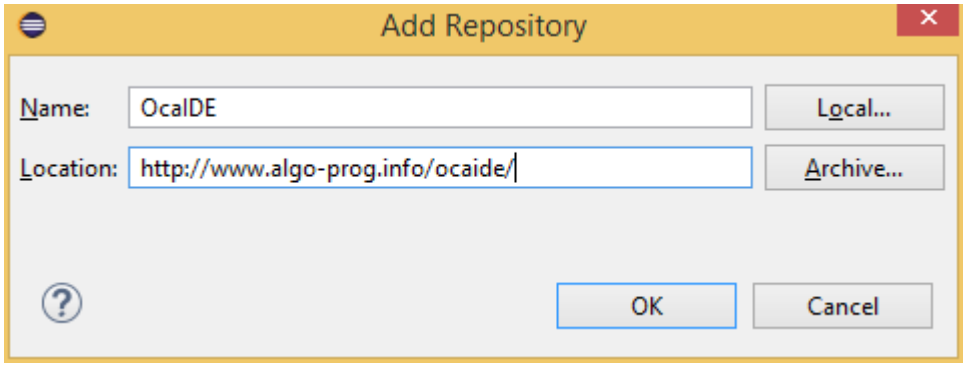
2.3 Instalação do ambiente interativo no Eclipse

Para instalar OcaIDE, um plugin para Eclipse:

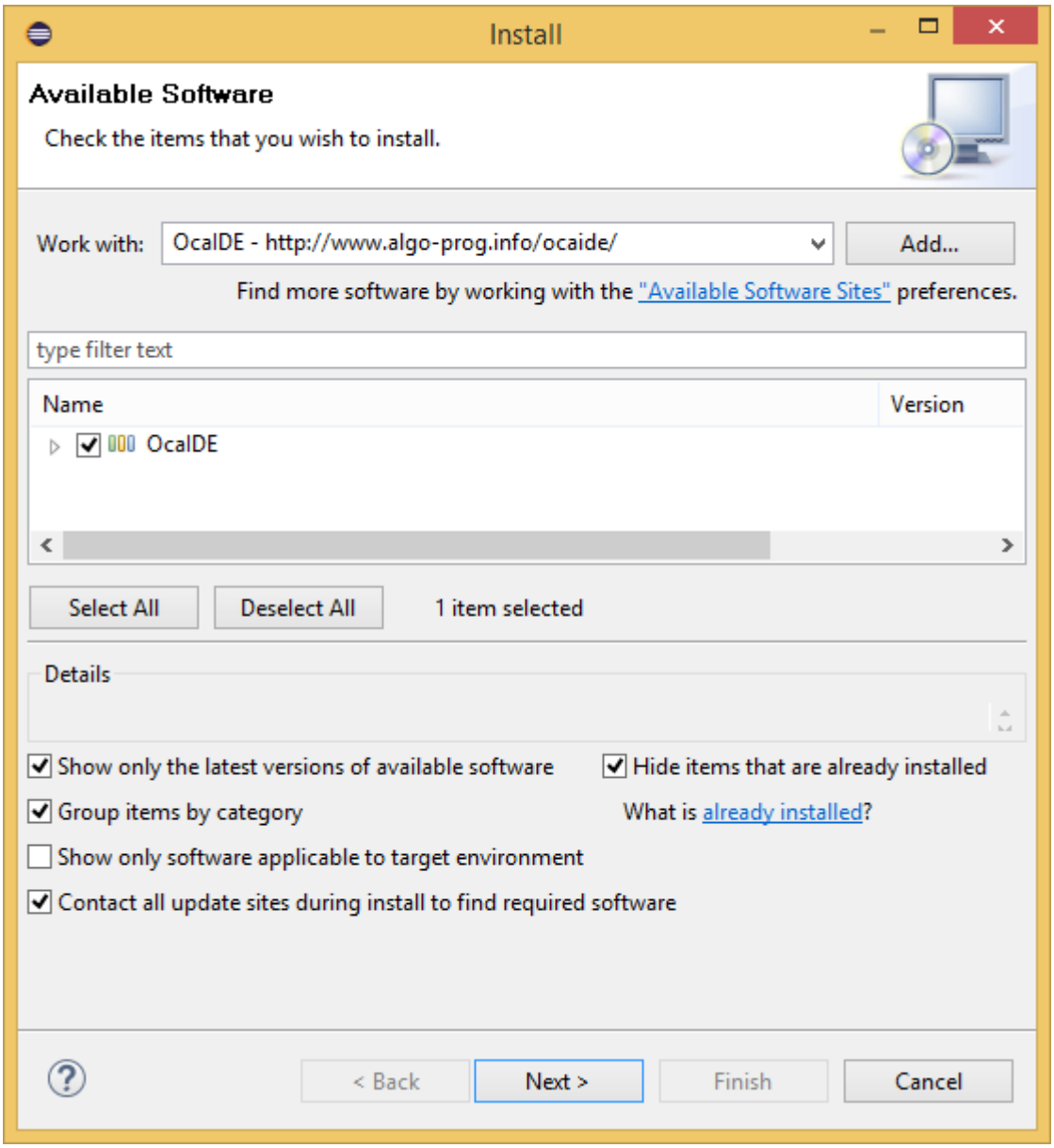
1. Instale a máquina virtual do Java versão 1.7 ou superior (`http://java.com/en/download/index.jsp`).
2. Instale uma versão recente do Eclipse (`http://www.eclipse.org/downloads/`).
3. Inicie o Eclipse.
4. No Eclipse acesse o menu `Help`→`Install New Software...`.



5. Clique Add para configurar um novo repositório.
6. No campo Name digite `OcaIDE` e no campo Location digite `http://www.algo-prog.info/ocaide/`.



7. Clique em Ok.
8. Selecione a categoria OcaIDE na lista de plugins.



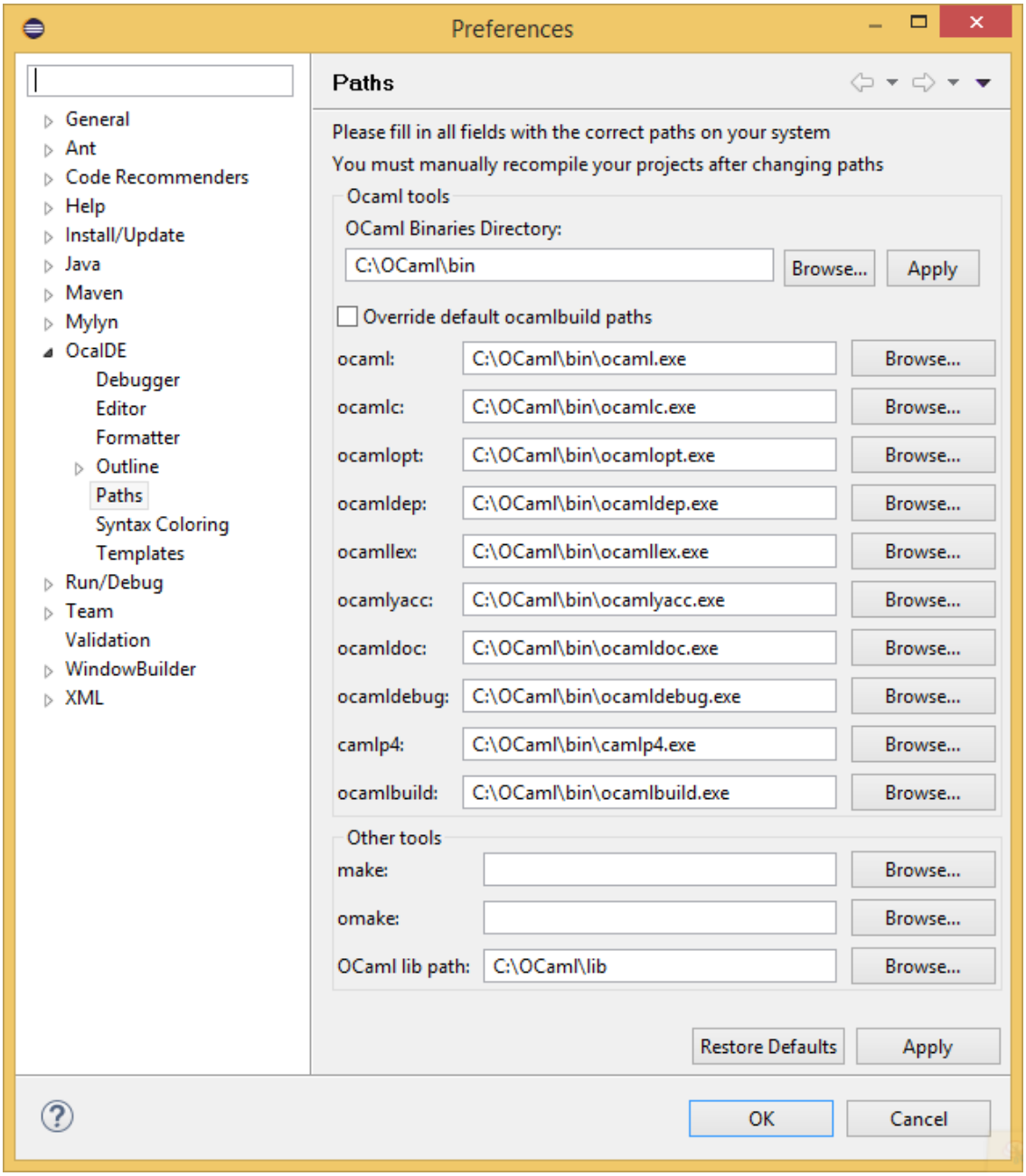
9. Clique Next.
10. Clique Next.
11. Aceite os termos do acordo de licença.
12. Clique Finish. Será feito o download do plugin.
13. Aceite a instalação (o plugin não tem assinatura digital).

14. Reinicie o Eclipse quando solicitado.

15. Após o reinício do Eclipse, o ambiente já está pronto para uso.

16. Para acessar a ajuda online clique no menu `Help→Help Contents` e escolha *OCaml Development User Guide* na lista de tópicos disponíveis.

17. Verifique os caminhos onde o OCaml pode ser encontrado. Para tanto acesse o menu `Window→Preferences` e selecione a aba `OcaIDE→Paths`. Se necessário preencha o campo `Ocaml Binaires Directory` para inidicar a localização correta dos programas executáveis do OCaml e em seguida clique em `Apply`. Pode ser necessário indicar também o caminho das bibliotecas do OCaml no campo `Ocaml lib path`. Conclua clicando em `OK`.



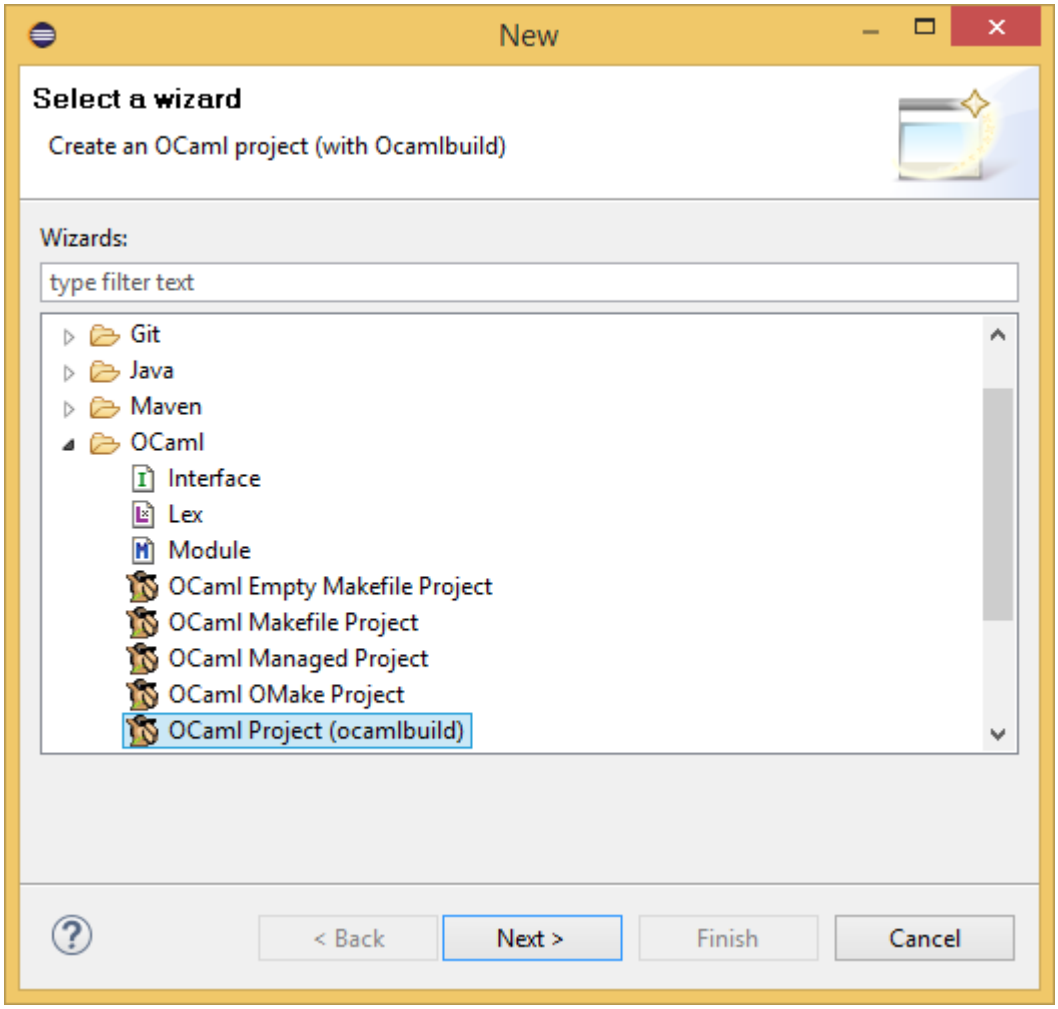
3 Desenvolvimento do primeiro projeto

Vamos desenvolver uma pequena aplicação para exibir uma mensagem na tela, usando o Eclipse.

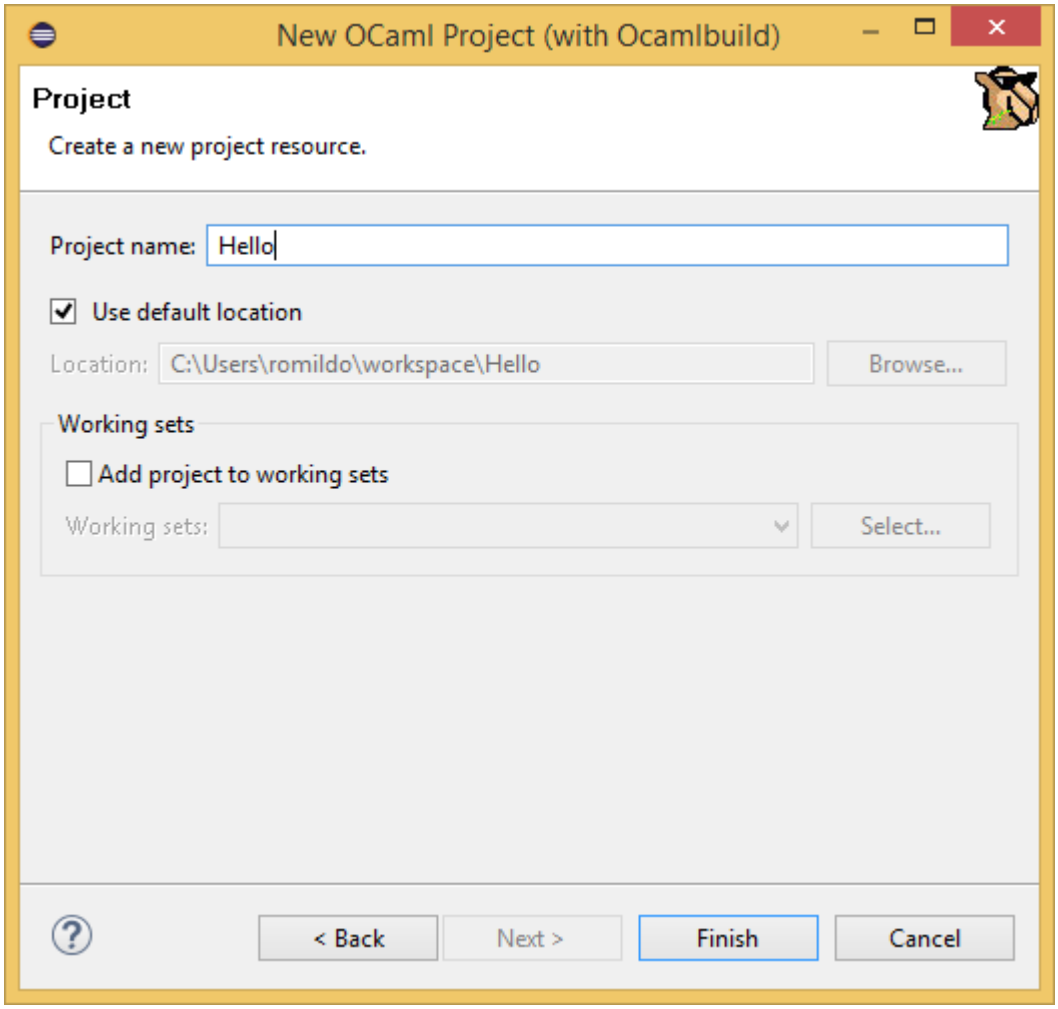
1. Para criar um novo projeto em OCaml no Eclipse use o menu `File→New→Other`.

2. Escolha a opção `OCaml Project (ocamlbuild)` na janela que se abre.

3. Em seguida clique em `Next`.

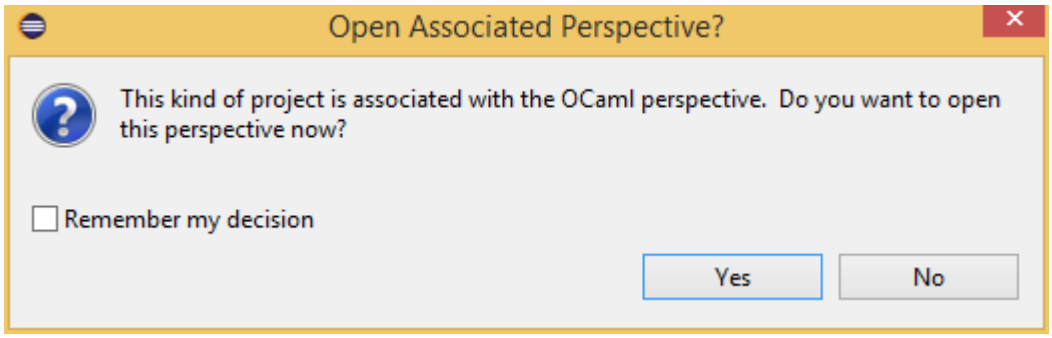


4. Digite o nome do projeto.



5. Clique em `Finish`.

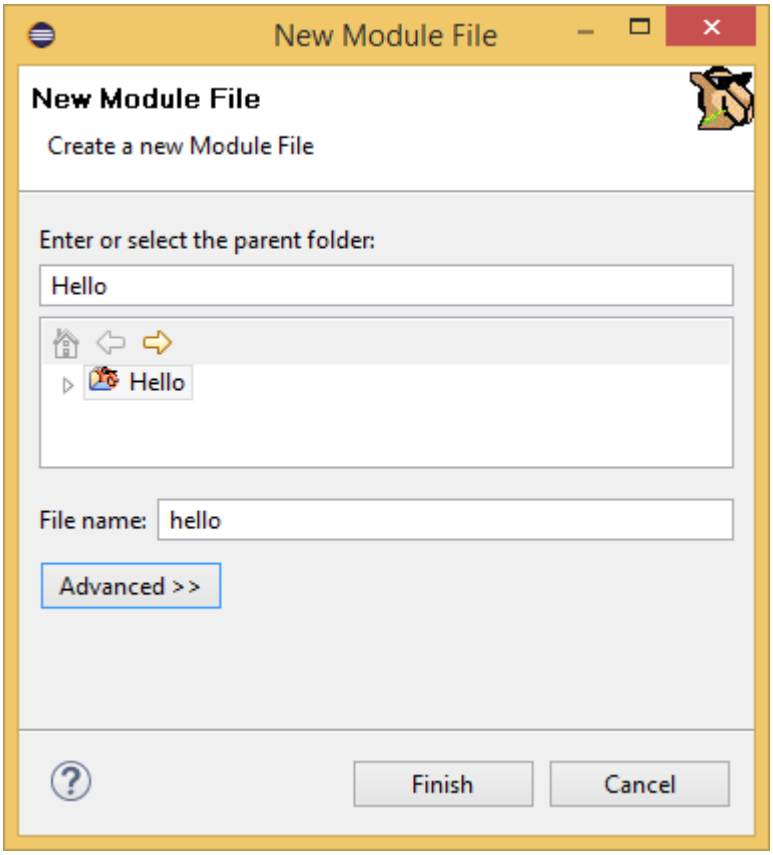
6. Aceite abrir a perspectiva OCaml associada ao projeto clicando em `Yes`.



7. Clique com o botão direito do mouse sobre o nome do projeto na visão `Navigator`. Será exibido um menu.

8. Escolha a opção `New`→`Module`. Será exibido o assistente `New Module File`.

9. No campo `File name` digite o nome do arquivo `hello`.



10. Clique em `Finish`. O módulo será adicionado ao projeto e aberto no editor de texto.

11. Vamos agora configurar algumas propriedades do projeto. Clique novamente com o botão direito do mouse sobre o nome do projeto na visão `Navigator`. Será exibido um menu.

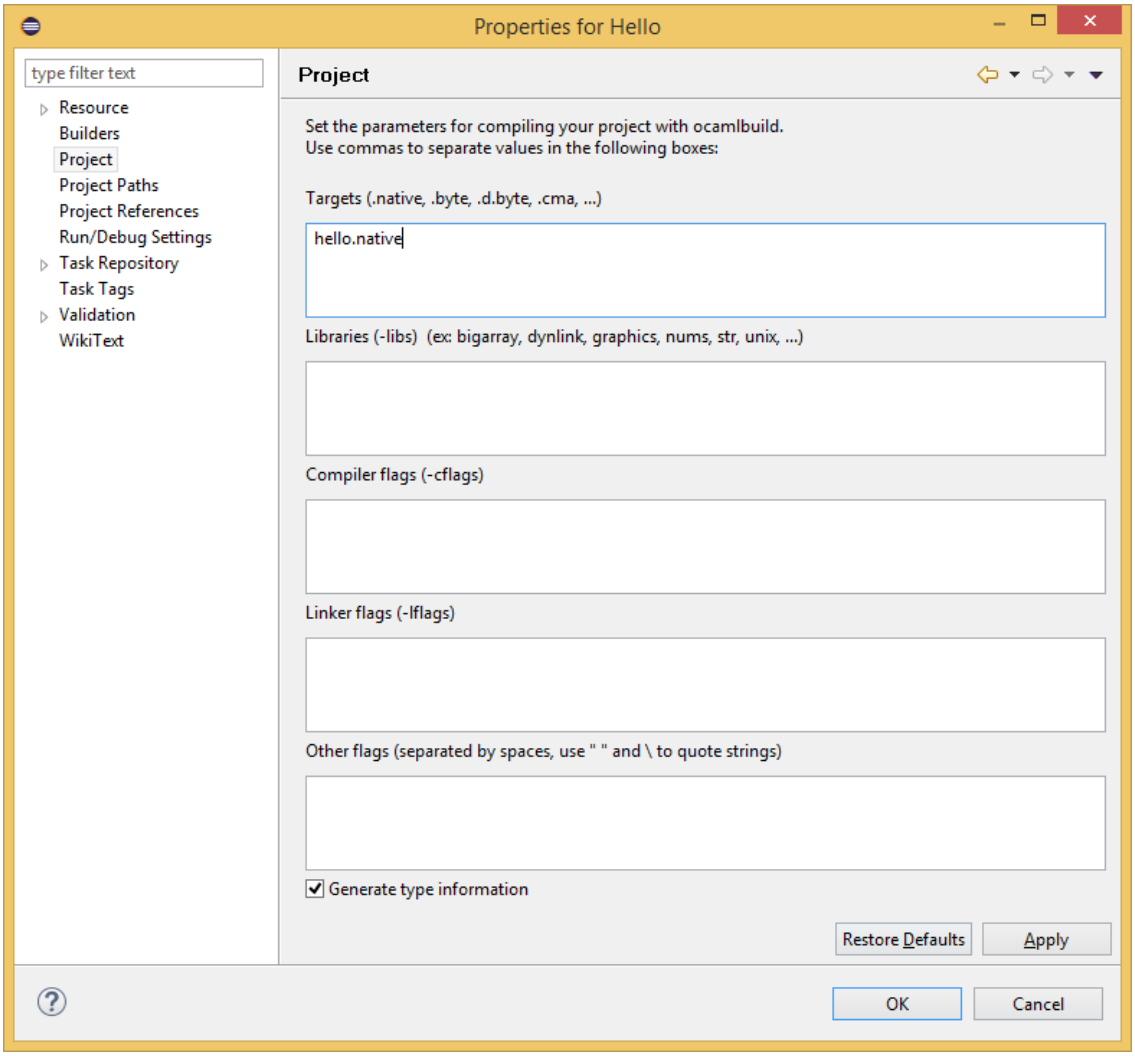
12. Escolha a opção `Properties`. Será exibido o assistente `Properties for Hello`.

13. Clique na opção `Project`.

14. No campo `Targets (.native, .byte, .d.byte, .cma, ...)` digite os alvos a serem construídos pelo projeto. Algumas possibilidades são:

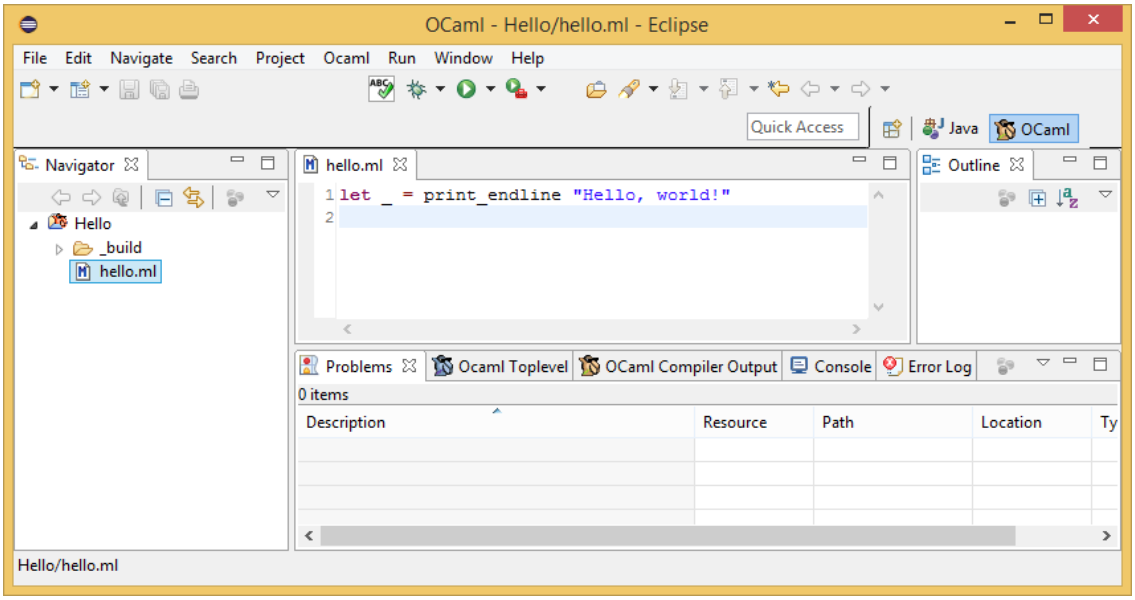
    - `hello.native`: programa executável nativo
    - `hello.byte`: programa executável em bytecodes
    - `hello.d.byte`: programa executável em bytecodes com código adicional para depuração

    Escolha `hello.native`.



15. Clique em `OK`.

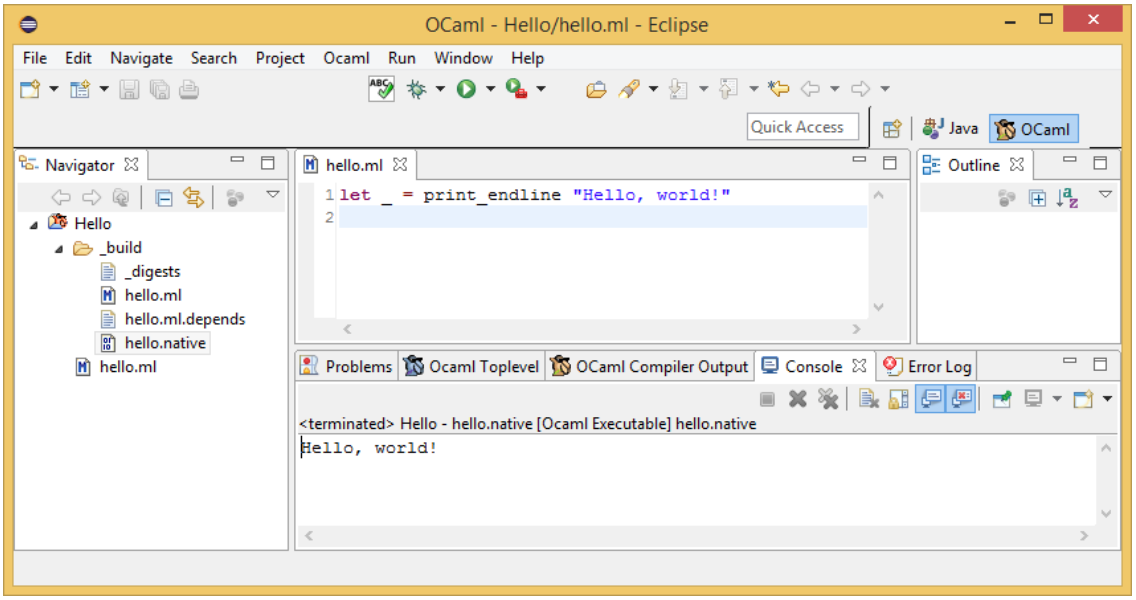
16. Digite o texto do programa fonte no arquivo do módulo usando o editor de texto.



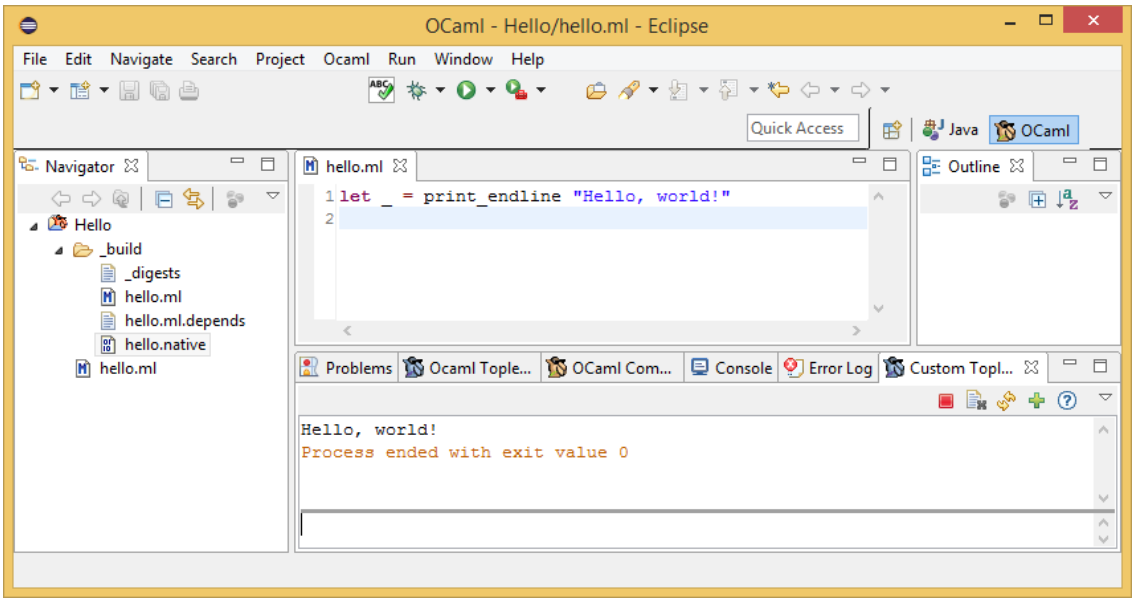
17. Salve o arquivo (menu `File→Save`). O módulo será automaticamente compilado.

18. Para executar o programa, clique com o botão direito do mouse sobre o nome do módulo executável na visão `Navigator`: `Hello→_build→hello.native`. Será exibido um menu.

19. Escolha a opção `Properties→Run As→Ocaml Executable` para executar o módulo na visão `Console`.



ou escolha a opção `Properties→Run As→Ocaml Toplevel` para carregar o módulo na visão `Custom Toplevel`.



4 Atividades

Tarefa 1.1: Produto de três números inteiros

Escreva um programa que solicita ao usuário três números inteiros, lê os números, e calcula e exibe o produto dos números.

Tarefa 1.2: Conversão de temperatura

Escreva um programa em OCaml que solicita ao usuário uma temperatura na escala Fahrenheit, lê esta temperatura, converte-a para a escala Celsius, e exibe o resultado.

Para fazer a conversão, defina uma função `celsius : float -> float` que recebe a temperatura na escala Fahrenheit e resulta na temperatura correspondente na escala Celsius. Use a seguinte equação para a conversão:

$$C = \frac{5}{9} \times (F - 32)$$

onde F é a temperatura na escala Fahrenheit e C é a temperatura na escala Celsius.

Use a função `celsius` na definição de uma função principal chamada `main`.

Tarefa 1.3: Peso ideal

Escrever um programa em OCaml que recebe a altura e o sexo de uma pessoa e calcula e mostra o seu peso ideal, utilizando as fórmulas constantes na tabela a seguir.

sexo	**peso ideal**
masculino	$72.7 \times h - 58$
feminino	$62.1 \times h - 44.7$

onde h é a altura da pessoa.

Tarefa 1.4: Situação de um aluno

Faça um programa que receba três notas de um aluno, e calcule e mostre a média aritmética das notas e a situação do aluno, dada pela tabela a seguir.

média das notas	**situação**
menor que 3	reprovado
entre 3 (inclusive) e 7	exame especial
acima de 7 (inclusive)	aprovado

Tarefa 1.5: Raízes da equação do segundo grau

Faça um programa que leia os coeficientes de uma equação do segundo grau e calcule e mostre suas raízes reais, caso existam.

Tarefa 1.6: Linha de crédito

A prefeitura de Contagem abriu uma linha de crédito para os funcionários estatutários. O valor máximo da prestação não poderá ultrapassar 30% do salário bruto.

Fazer um programa que permita entrar com o salário bruto e o valor da prestação, e informar se o empréstimo pode ou não ser concedido.

Tarefa 1.7: Classe eleitoral

Crie um programa que leia a idade de uma pessoa e informe a sua classe eleitoral:

não eleitor abaixo de 16 anos;

eleitor obrigatório entre 18 (inclusive) e 65 anos;

eleitor facultativo de 16 até 18 anos e acima de 65 anos (inclusive).

Tarefa 1.8: Impostos

Faça um programa que apresente o menu a seguir, permita ao usuário escolher a opção desejada, receba os dados necessários para executar a operação, e mostre o resultado.

```
--------------------------------
Opções:
--------------------------------
1. Imposto
2. Novo salário
3. Classificação
--------------------------------
Digite a opção desejada:
```

Verifique a possibilidade de opção inválida.

Na **opção 1** receba o salário de um funcionário, calcule e mostre o valor do imposto sobre o salário usando as regras a seguir:

salário	**taxa de imposto**
Abaixo de R$500,00	5%
De R$500,00 a R$850,00	15%
Acima de R$850,00	10%

Na **opção 2** receba o salário de um funcionário, calcule e mostre o valor do novo salário, usando as regras a seguir:

salário	**aumento**
Acima de R$1.500,00	R$25,00
De R$750,00 (inclusive) a R$1.500,00 (inclusive)	R$50,00
De R$450,00 (inclusive) a R$750,00	R$75,00
Abaixo de R$450,00	R$100,00

Na **opção 3** receba o salário de um funcionário e mostre sua classificação usando a tabela a seguir:

salário	**classificação**
Até R$750,00 (inclusive)	mal remunerado
Acima de R$750,00	bem remunerado

Tarefa 1.9: Terno pitagórico

Em Matemática um **terno pitagórico** (ou trio pitagórico, ou ainda tripla pitagórica) é formado por três números naturais a, b e c tais que $a^2 + b^2 = c^2$. O nome vem do *teorema de Pitágoras* que afirma que se as medidas dos lados de um triângulo retângulo são números inteiros, então elas formam um terno pitagórico.

Codifique um programa que leia três números naturais e verifique se eles formam um terno pitagórico.

Tarefa 1.10: Palíndromes

Escreva um programa em OCaml que solicita ao usuário para digitar uma frase, lê a frase (uma linha) da entrada padrão e testa se a string lida é uma palíndrome, exibindo uma mensagem apropriada.

5 Soluções

Tarefa 1.1 on page 1-11: Solução

```
(* Produto de três números inteiros *)

let main () =
  try
    print_string "Digite um número inteiro: ";
    let x = read_int () in
    print_string "Digite outro número inteiro: ";
    let y = read_int () in
    print_string "Digite outro número inteiro: ";
    let z = read_int () in
    let p = x * y * z in
    print_newline ();
    print_string "Produto dos números digitados: ";
    print_int p;
    print_newline ()
  with
    Failure "int_of_string" -> print_endline "Dados incorretos"

let _ = main ()
```

Tarefa 1.2 on page 1-11: Solução

```
(* conversão de temperaturas *)

open Printf

let celsius f =
  5.0 /. 9.0 *. (f -. 32.0)

let main () =
  print_string "Digite a temperatura em F: ";
  let temp = read_float () in
  printf "%.2f F equivale a %.2f F\n" temp (celsius temp)

let _ = main ()
```

Tarefa 1.3 on page 1-12: Solução

```
(* Cálculo do peso ideal *)

let main () =
  try
    print_endline "peso ideal";
    print_endline "---------";
    print_string "altura (em metros): ";
    let altura = read_float () in
    print_string "sexo (m/f): ";
    let sexo = String.lowercase (String.trim (read_line ())) in
    if sexo = "m" then
      Printf.printf "peso ideal: %.2f Kg\n" (72.7 *. altura -. 58.0)
    else if sexo = "f" then
      Printf.printf "peso ideal: %.2f Kg\n" (62.1 *. altura -. 44.7)
    else
      print_endline "sexo inválido"
  with
    Failure _ -> print_endline "dados inválidos"

let _ = main ()
```

Tarefa 1.4 on page 1-12: Solução

```
(* Cálculo do peso ideal *)

let main () =
  try
    print_endline "peso ideal";
    print_endline "---------";
    print_string "altura (em metros): ";
    let altura = read_float () in
    print_string "sexo (m/f): ";
    let sexo = String.lowercase (String.trim (read_line ())) in
    if sexo = "m" then
      Printf.printf "peso ideal: %.2f Kg\n" (72.7 *. altura -. 58.0)
    else if sexo = "f" then
      Printf.printf "peso ideal: %.2f Kg\n" (62.1 *. altura -. 44.7)
    else
      print_endline "sexo inválido"
  with
    Failure _ -> print_endline "dados inválidos"

let _ = main ()
```

Tarefa 1.5 on page 1-12: Solução

```
(* solução da equação do segundo grau *)

let main () =
  try
    print_endline "digite os coeficientes da equação do segundo grau:";
    print_string "a: ";
    let a = read_float () in
    if a = 0.0 then
      print_endline "a equação não é do segundo grau"
    else
      begin
        print_string "b: ";
        let b = read_float () in
        print_string "c: ";
        let c = read_float () in
        let delta = b *. b -. 4.0 *. a *. c in
        if delta < 0.0 then
          print_endline "a equação não possui raízes reais"
        else if delta = 0.0 then
          let r = -. b /. (2.0 *. a) in
          Printf.printf "raíz: %f\n" r
        else
          let r1 = (-. b +. sqrt delta) /. (2.0 *. a)
          and r2 = (-. b -. sqrt delta) /. (2.0 *. a) in
          Printf.printf "raízes: %f e %f\n" r1 r2
      end
  with
    Failure "float_of_string" -> print_endline "dados inválidos"

let _ = main ()
```

Tarefa 1.6 on page 1-12: Solução

```
(* linha de crédito *)

let _ =
  try
    print_endline "Linha de crédito";
    print_endline "======================";
    print_string "salário bruto: ";
    let sal_bruto = read_float () in
    print_string "valor da prestação: ";
    let valor_prestacao = read_float () in
    if valor_prestacao > 0.3 *. sal_bruto then
      print_endline "o empréstimo não pode ser concedido"
    else
      print_endline "o empréstimo pode ser concedido"
  with
    Failure "float_of_string" -> print_endline "dados inválidos"
```

Tarefa 1.7 on page 1-12: Solução

```
let _ =
  try
    print_endline "Classe eleitoral";
    print_endline "================";
    print_string "idade do eleitor: ";
    let idade = read_int () in
    if idade < 16 then
      print_endline "não eleitor"
    else if idade >= 18 && idade < 65 then
      print_endline "eleitor obrigatório"
    else
      print_endline "eleitor facultativo"
  with
    Failure "int_of_string" -> print_endline "dados inválidos"
```

Tarefa 1.8 on page 1-13: Solução

```
let imposto () =
  print_endline "Imposto";
  print_string "salário: ";
  let salario = read_float () in
  let taxa =
    if salario < 500.0 then
      5.0
    else if salario < 850.0 then
      10.0
    else
      15.0
  in
  let imp = salario *. taxa /. 100.0 in
  Printf.printf "imposto: %.2f\n" imp

let novo_salario () =
  print_endline "Novo salário";
  print_string "salário: ";
  let salario = read_float () in
  let aumento =
    if salario > 1500.0 then
      25.0
    else if salario >= 750.0 then
      50.0
    else if salario >= 450.0 then
      75.0
    else
      100.0
  in
  Printf.printf "novo salário: %.2f\n" (salario +. aumento)

let classificacao () =
  print_endline "Classificação";
  print_string "salário: ";
  let salario = read_float () in
  if salario <= 750.0 then
    print_endline "mal remunerado"
  else
    print_endline "bem remunerado"

let main () =
  try
    print_endline "-----------------------------";
    print_endline "Opções:";
    print_endline "-----------------------------";
    print_endline "1. Imposto";
    print_endline "2. Novo salário";
    print_endline "3. Classificação";
    print_endline "-----------------------------";
    print_string "Digite a opção desejada: ";
    let opcao = read_int () in
    match opcao with
    | 1 -> imposto ()
    | 2 -> novo_salario ()
    | 3 -> classificacao ()
    | _ -> print_endline "Opção inválida"
  with
    Failure _ -> print_endline "dados inválidos"

let _ = main ()
```

Tarefa 1.9 on page 1-13: Solução

```
let _ =
  try
    print_endline "Verificação de ternos pitagóricos";
    print_string "digite o primeiro número: ";
    let a = read_int () in
    print_string "digite o segundo número: ";
    let b = read_int () in
    print_string "digite o terceiro número: ";
    let c = read_int () in
    if a*a = b*b + c*c || b*b = a*a + c*c || c*c = a*a + b*b then
      print_endline "Os números formam um terno pitagórico"
    else
      print_endline "Os números não formam um terno pitagórico"
  with
    Failure "int_of_string" -> print_endline "dados inválidos"
```

Tarefa 1.10 on page 1-13: Solução

```
let palindrome str =
  let n = String.length str in
  let rec loop i =
    if i = n/2 then
      true
    else
      if str.[i] = str.[n - i - 1] then
        loop (i+1)
      else
        false
  in
  loop 0

let main () =
  print_endline "Digite uma frase:";
  let str = read_line () in
  if palindrome str then
    print_endline "é palíndrome"
  else
    print_endline "não é palíndrome"

let _ = main ()
```

2 OCaml: Recursividade

Resumo

Neste capítulo você se familiarizará com a definição de funções recursivas em OCaml.

Sumário

1 Atividades

Tarefa 2.1: Fatorial

Escrever um programa para calcular o fatorial de um número natural, dado por

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n(n-1)! & \text{if } n > 0 \end{cases}$$

O número deve ser dado como argumento da linha de comando.

Tarefa 2.2: Média

Escrever um programa para calcular a média das notas obtidas pelos alunos de uma turma em uma avaliação. A entrada das notas deve ser encerrada com um valor inválido para a nota, isto é, uma nota negativa.

Tarefa 2.3: Correção de provas de múltipla escolha

Vamos escrever um programa em OCaml para corrigir provas de múltipla escolha que foram aplicadas para uma turma de alunos.

Os dados de entrada deverão ser obtidos de um arquivo texto. O nome deste arquivo deve ser informado como um argumento da linha de comando. O arquivo de entrada deve conter:

- o gabarito (as respostas corretas de cada questão) da prova: primeira linha do arquivo de entrada
- a matrícula e as respostas de cada aluno da turma: demais linhas do arquivo de entrada

Exemplo de entrada:

```
      abcacdaeacab
10014 abcbcbeeacad
10310 bbaaadaeacab
40010 aaaaaaaaaaaa
20101 abdecbbeaaab
20200 cbdbceaeacab
20330 abcbdbaeacab
30355 ebecceaeadad
```

As notas devem ser normalizadas na faixa de zero a dez. Assim para calcular a nota obtida em uma prova, divida a soma dos pontos obtidos (um ponto para cada resposta correta) pelo número de questões na prova, e multiplique o resultado por dez.

Um aluno será aprovado se e somente se sua nota for igual ou superior a sete.

O programa deve calcular e mostrar:

- a matrícula, a nota e a situação (aprovado ou reprovado) de cada aluno, e
- a taxa (em porcentagem) de aprovação.

2 Soluções

Tarefa 2.1 on page 2-1: Solução

```
(* fatorial *)

let rec fat n =
  if n = 0 then
    1
  else
    n * fat (n - 1)

let main () =
  if Array.length Sys.argv = 2 then
    try
      let num = int_of_string Sys.argv.(1) in
      print_int (fat num);
      print_newline ()
    with
      Failure "int_of_string" -> exit 2
  else
    exit 1

let _ = main ()
```

3 Estruturas de Dados

Resumo

Neste capítulo vamos conhecer algumas estruturas de dados disponíveis no OCaml.

A maioria das estruturas de dados em OCaml são imutáveis, ou seja, não podem ser alteradas. Algumas poucas são mutáveis, como por exemplo os arrays.

Sumário

1 Tuplas

Uma tupla é uma sequência de valores de tamanho fixo. Os componentes da tupla não precisam ter o mesmo tipo.

O tipo de uma tupla é o tipo produto, indicado usando o construtor de tipos `*`. Exemplos:

tipo	comentário
`int * string`	tipo dos pares de inteiro e string
`float * bool`	tipo dos pares de real e booleano
`float * bool * string`	tipo das triplas de real, booleano e string

Uma tupla é escrita sepando os seus componentes por uma vírgula (`,`). Usualmente escreve-se a tupla entre parênteses, embora os parênteses não façam parte da sintaxe de tuplas em OCaml. Parênteses podem ser usados para agrupar subexpressões dentro de expressões maiores. Exemplos:

tipo	tipo	comentário
`2,'h'`	`int * char`	
`"Maria", 10, true, 154.6`	`string * int * bool * float`	
`(10, 5.8)`	`int * float`	

O construtor de tupla pode ser usado tanto para criar novas tuplas, como também em um padrão para decompor uma tupla em seus componentes.

Algumas operações com tuplas:

operação	tipo	comentário
`fst`	`'a * 'b -> 'a`	seleciona o primeiro elemento de um par
`snd`	`'a * 'b -> 'b`	seleciona o segundo elemento de um par

Tuplas são semelhantes a registros (estruturas em C), porém os seus componentes são identificados pela posição que ocupam na tupla, e não por um nome de campo.

2 Listas

Uma lista é uma sequência de valores do mesmo tipo e podem ser de qualquer tamanho.
O tipo de uma lista é indicado usando o construtor de tipos `list`. Exemplos:

tipo	comentário
`int list`	tipo das listas de inteiros
`float list`	tipo das listas de reais
`char list`	tipo das listas de caracteres
`string list`	tipo das listas de string
`int list list`	tipo das listas de listas de inteiros
`(int * string) list`	tipo das listas de pares de inteiro e string
`(string -> int) list`	tipo das listas de funções de string em inteiro
`'a list`	tipo de todas as listas

Uma lista pode ser escrita colocando os seus elementos entre colchetes (`[` e `]`) e separando-os por ponto-e-vírgula (`;`). Exemplos:

tipo	tipo	comentário
`[2; 3 ; 56; 0]`	`int list`	
`['A'; 'n'; 'a']`	`char list`	
`[]`	`'a list`	a lista vazia

Uma lista pode ser de duas formas possíveis:

- lista vazia
  - `[]`
  - não tem nenhum elemento
- lista não vazia
  - *x* `::` *xs*
  - tem uma cabeça (primeiro elemento) e uma cauda (lista dos demais elementos)
  - o operador `::` associa-se à direita

Os construtores `[]` e `::` podem ser usados tanto para criar novas listas, como também em um padrão para decompor uma lista em seus componentes. Exemplos:

lista	notação básica	comentário
`[]`	`[]`	a lista vazia
`['A']`	`'A' :: []`	
`[10; 20]`	`10 :: 20 :: []`	

Algumas operações com listas:

operação	tipo	comentário
`@`	`'a list -> 'a list -> 'a list`	concatenação de listas
`List.length`	`'a list -> int`	número de elementos da lista
`List.hd`	`'a list -> 'a`	seleciona a cabeça da lista
`List.tl`	`'a list -> 'a`	seleciona a cauda da lista
`List.nth`	`'a list -> int -> 'a`	seleciona o *n*-ésimo elemento da lista, contando a partir de 0
`List.rev`	`'a list -> 'a list`	inverte os elementos

3 Opções

Uma opção é usada para expressar que um valor pode ou não estar presente.
O tipo de uma opção é indicado usando o construtor de tipos `option`. Exemplos:

tipo	comentário
`int option`	tipo das opções de inteiro
`float option`	tipo das opções de real
`char option`	tipo das opções de caracter
`string option`	tipo das opções de string
`(int * string) option`	tipo das opções de par de inteiro e string
`(string -> int) option`	tipo das opções de função de string em inteiro

Uma opção pode ser de duas formas possíveis:

- opção vazia
  - `None`
  - não tem nenhum elemento
- opção não vazia
  - `Some x`
  - tem um elemento

Os construtores `None` e `Some` podem ser usados tanto para criar novas opções, como também em um padrão para decompor uma opção em seus componentes. Exemplos:

opção	comentário
`None`	a opção vazia
`Some 'A'`	
`Some [10; 20]`	

4 Arrays

Um array é uma sequência de valores do mesmo tipo e podem ser de qualquer tamanho. Os elementos de um array são alocados em posições consecutivas de memória e são indexados eficientemente por um inteiro que indica sua posição na sequência.

O tipo de um array é indicado usando o construtor de tipos `array`. Exemplos:

tipo	comentário
`int array`	tipo dos arrays de inteiros
`float array array`	tipo dos arrays de arrays de reais
`string array`	tipo dos arrays de string
`int list array`	tipo dos arrays de listas de intaeiros
`(int * string) array`	tipo dos arrays de pares de inteiro e string
`(string -> int) array`	tipo dos arrays de funções de string em inteiro

Um array pode ser escrita colocando os seus elementos entre colchetes e barra duplos (`[|` e `|]`) e separando-os por ponto-e-vírgula (`;`). Exemplos:

tipo	tipo	comentário
`[|2; 3 ; 56; 0|]`	`int array`	
`[|'A'; 'n'; 'a'|]`	`char array`	
`[||]`	`'a array`	a array vazia

Algumas operações com arrays:

operação	tipo	comentário
`Array.length`	`'a array -> int`	número de elementos do array

5 Casamento de padrão

Um padrão especifica a estrutura de um valor e pode ser usado na operação casamento de padrão.

O casamento de um padrão e um valor pode suceder ou falhar. Em caso de sucesso novas variáveis podem ser instanciadas.

5.1 Padrão constante

O **padrão constante** é simplesmente uma *constante*. O **casamento** *sucede* se e somente se o padrão for *idêntico* ao valor. Nenhuma associação de **variável** é produzida.

Exemplos:

padrão	valor	casamento
`10`	`10`	✓
`10`	`28`	×
`10`	`'P'`	*erro de tipo*
`'P'`	`'P'`	✓
`'P'`	`'q'`	×
`'P'`	`true`	*erro de tipo*
`true`	`true`	✓
`true`	`false`	×
`true`	`65`	*erro de tipo*

✓: sucede

×: falha

5.2 Padrão variável

O **padrão variável** é simplesmente um identificador de *variável* de valor (e como tal deve começar com letra minúscula). O **casamento** *sucede sempre*. A **variável** é associada ao *valor*.

Exemplos:

padrão	valor	casamento
`x`	`10`	✓ `x` ↦ `10`
`alfa`	`563.1223`	✓ `alfa` ↦ `563.1223`
`letra`	`'K'`	✓ `letra` ↦ `'K'`
`nome_cliente`	`"Ana Maria"`	✓ `nome_cliente` ↦ `"Ana Maria"`
`pessoa`	`("Ana",'F',16)`	✓ `pessoa` ↦ `("Ana",'F',16)`
`notas`	`[5.6;7.1;9.0]`	✓ `notas` ↦ `[5.6;7.1;9.0]`

5.3 Padrão curinga

O **padrão curinga** é escrito como um sublinhado (_). O **casamento** *sucede* sempre. *Nenhuma* associação de **variável** é produzida. _ é também chamado de **variável anônima**, pois casa com qualquer valor sem dar nome ao valor.

Exemplos:

padrão	valor	casamento
_	`10`	✓
_	`28`	✓
_	`'P'`	✓
_	`()`	✓
_	`(18,3,2012)`	✓
_	`"Ana Maria"`	✓
_	`[5.6;7.1;9.0]`	✓

5.4 Padrão com construtor

Um construtor de dados pode ser usado em um padrão. Subpadrões são usados para indicar os componentes da estrutura de dados.

O **casamento** *sucede* se e somente se os construtores no padrão e no valor forem os mesmos, e os subpadrões e os valores componentes da estrutura de dados casarem. Se o tipo do padrão e o tipo do valor não forem compatíveis ocorre um erro de tipo.

Exemplos de padrão tupla:

padrão	valor	casamento
`(18,true)`	`(18,true)`	✓
`(97,true)`	`(18,true)`	×
`(18,false)`	`(18,true)`	×
`(18,'M')`	`(18,true)`	*erro de tipo*
`(18,true,'M')`	`(18,true)`	*erro de tipo*
`()`	`()`	✓
`(x,y)`	`(5,9)`	✓ `x` ↦ `5`, `y` ↦ `9`
`(d,_,a)`	`(5,9,2012)`	✓ `d` ↦ `5`, `a` ↦ `2012`
`(x,y,z)`	`(5,9)`	*erro de tipo*
`(18,m,a)`	`(18,3,2012)`	✓ `m` ↦ `3`, `a` ↦ `2012`
`(d,5,a)`	`(18,3,2012)`	×
`(nome,sexo,_)`	`("Ana",'F',18)`	✓ `nome` ↦ `"Ana"`, `sexo` ↦ `'F'`
`(_,_,idade)`	`("Ana",'F',18)`	✓ `idade` ↦ `18`
`(_,(_,fam),9)`	`('F',("Ana","Dias"),9)`	✓ `fam` ↦ `"Dias"`
`(_,(_,fam),5)`	`('F',("Ana","Dias"),9)`	×

Exemplos com padrões lista:

- O padrão `[]` casa somente com a lista vazia.

padrão	valor	casamento
`[]`	`[]`	✓
`[]`	`[1;2;3]`	×

- O padrão `x::xs` casa com qualquer lista não vazia, associando as variáveis `x` e `xs` com a cabeça e a cauda da lista, respectivamente.

padrão	valor	casamento
`x::xs`	`[]`	×
`x::xs`	`[1;2;3;4]`	✓ `x` ↦ `1`, `xs` ↦ `[2;3;4]`
`x::xs`	`['A']`	✓ `x` ↦ `'A'`, `xs` ↦ `[]`

- O padrão `x::y::_` casa com qualquer lista que tenha pelo menos dois elementos, associando as variáveis `x` e `y` ao primeiro e segundo elementos da lista, respectivamente.

padrão	valor	casamento
`x::y::_`	`[]`	×
`x::y::_`	`["ana"]`	×
`x::y::_`	`[1;2]`	✓ `x` ↦ `1`, `y` ↦ `2`
`x::y::_`	`[1;2;3;4]`	✓ `x` ↦ `1`, `y` ↦ `2`

- O padrão `x::_::z::[]` casa com qualquer lista que tenha exatamente três elementos, associando as variáveis `x` e `z` ao primeiro e terceiro elementos da lista, respectivamente.

padrão	valor	casamento
`x::_::z::[]`	`[]`	×
`x::_::z::[]`	`["ana"]`	×
`x::_::z::[]`	`[1;2;3]`	✓ `x` ↦ `1`, `z` ↦ `3`
`x::_::z::[]`	`[1;2;3;4;5]`	×

- O padrão `0::a::_` casa com qualquer lista de números que tenha pelo menos dois elementos, sendo o primeiro igual a zero, e associando a variável `a` ao segundo elemento da lista.

padrão	valor	casamento
`0::a::_`	`[]`	×
`0::a::_`	`[0]`	×
`0::a::_`	`[0;2;3]`	✓ `a` ↦ `2`
`0::a::_`	`[0;10;6;3]`	✓ `a` ↦ `10`
`0::a::_`	`[7;0;8]`	×

- O padrão `(m,_)::_` casa com qualquer lista não vazia de pares, associando a variável `m` ao primeiro componente do par que é o primeiro elemento da lista.

padrão	valor	casamento
`(m,_)::_`	`[]`	×
`(m,_)::_`	`[("fim",true)]`	✓ `m` ↦ `"fim"`
`(m,_)::_`	`[(10,'M');(20,'F')]`	✓ `m` ↦ `10`

O padrão `[1;alfa]` casa com qualquer lista de dois números que começa com `1`, associando a variável `alfa` ao segundo elemento da lista.

padrão	valor	casamento
`[1;alfa]`	`[]`	×
`[1;alfa]`	`[1]`	×
`[1;alfa]`	`[1;5]`	✓ `alfa` ↦ `5`
`[1;alfa]`	`[9;5]`	×
`[1;alfa]`	`[1;2;3]`	×

Exemplos com padrão opção:

padrão	valor	casamento
`None`	`None`	✓
`None`	`Some 56`	×
`Some 7`	`None`	×
`Some 7`	`Some 7`	✓
`Some peso`	`Some 58.6`	✓ `peso` ↦ `58.6`
`Some (_::x::_)`	`Some [1;3;5;7;9]`	✓ `x` ↦ `3`
`Some (_::x::_)`	`Some [1]`	×

5.5 Expressão match

A expressão match é da forma

match *expressão* with *match*

ond *match* é da forma

padrão$_1$ -> *expressão*$_1$ |
padrão$_2$ -> *expressão*$_2$ |
... |
padrão$_n$ -> *expressão*$_n$ |

Primeiramente a expressão incicial *expressão* é avaliada, e então o seu valor é comparado com cada um dos *padrões*. Quando um Padrão *padrão*$_1$ que casa é encontrado, a expressão correspondente *expressão*$_i$ é avaliada e o seu valor é o valor da expressão match.

Exemplos:

```
match n + 1 with
  1 -> "a"
| 2 -> "b"
| 3 -> "c"
| 4 -> "d"
| _ -> ""
```

```
match minhaLista with
| [] -> "empty"
| _::_ -> "nonempty"
```

5.6 Declaração de função

Em uma declaração de função os parâmetros formais são especificados usando padrões. Quando a função é aplicada acontece o casamento de padrão entre cada argumento e o respectivo padrão usado na definição. O corpo da função é avaliado em um ambiente estendido para conter as variáveis criadas pelo casamento de padrão.

Exemplo:

```
let dist (x1,y1) (x2,y2) =
  sqrt ((x1 -. x2)**2.0 +. (y1 -. y2)**2.0)
```

5.7 Funções anônimas

Uma função pode ser escrita sem definir um nome para ela.
Exemplos:

```
fun x -> 2*x
fun (x,y) -> (x+y)/2
```

Usando a palavra reservada **function** temos suporte a várias alternativas similares à uma expressão match.
Exemplos:

```
function x -> 2*x

function
  None -> 0
| Some x -> x

let rec soma = function
               | [] -> 0
               | x::xs -> x + soma xs
```

5.8 Declaração de valores

O lado esquerdo de uma declaração let para valores pode ser um padrão qualquer. A expressão é avaliada e então é feito o casamento de padrão entre o seu valor e o padrão, instanciando as variáveis que aparecem no padrão. Se o casamento de padrão falhar ocorre uma exceção.
Exemplos:

```
let x::_ = [10;20;30] in x + 1
```

Parte II

Introdução à Construção de Compiladores

4 Construindo um Interpretador

Resumo

Nesta aula vamos implementar um Interpretador para uma linguagem de programação bastante simples.

Ao desenvolver esta aplicação estaremos praticando a programação em OCaml e também nos familiarizando com a manipulação de estruturas de dados que representam programas.

Sumário

1 A linguagem *straight-line*

Straight-line é uma linguagem de programação muito simples apresentada por Andrew Appell em seu livro de compiladores para ilustrar alguns conceitos. O seu objetivo é apenas didático.

Um programa em *setraight-line* é um comando. Expressões simples podem aparecer dentro de alguns comandos.

A sintaxe de *straight-line* é definida pela gramática seguinte.

Stm	→	*Stm* `;` *Stm*	CompoundStm
Stm	→	`id :=` *Exp*	AssignStm
Stm	→	print `(` *ExpList* `)`	PrintStm
Exp	→	`id`	IdExp
Exp	→	`num`	NumExp
Exp	→	*Exp* *Binop* *Exp*	OpExp
Exp	→	`(` *Stm* `,` *Exp* `)`	EseqExp
ExpList	→	*Exp* `,` *ExpList*	PairExpList
ExpList	→	*Exp*	LastExpList
Binop	→	`+`	Plus
Binop	→	`-`	Minus
Binop	→	`*`	Times
Binop	→	`/`	Div

Veja um exemplo de um programa em *straight-line*:

```
a := 5+3;
b := ( print(a, a-1), 10*a );
print(b)
```

cuja saída é

```
8 7
80
```