Conception globale & modularité

I2 - Algorithmique et Programmation structurée

Julien Saunier, Alexandre Pauchet, Pierrick Tranouez

Plan

Conception globale

Rappel : Méthodologie générale de développement

La création d'un programme informatique suit 4 phases :

1. La spécification des besoins
   - Analyse et reformulation du problème posé
   - Production : **Cahier des charges**
2. **La conception globale**
   - Définition de l'architecture globale du programme
   - Production : **Analyse descendante**
3. La conception détaillée
   - Écriture des algorithmes de chaque sous-partie du programme
   - Production : **Algorithmes en pseudo-code**
4. L'implémentation
   - Transcription dans un langage de programmation
   - Production : **Programmes**

De façon orthogonale...

Génie logiciel

1. Comment spécifier ?
   - Cahier des charges, *user stories*...
2. Comment modéliser (conception globale) ?
   - Diagrammes d'usage, de fonctionnement...
3. Comment gérer la production de code ?
   - Répartition des tâches
4. Comment en assurer la qualité ?
   - Tests unitaires
   - Tests d'intégration

Modularité

Objectifs de la modularité 1 / 2

Définition (selon Wikipedia)

En informatique, la programmation modulaire consiste dans le regroupement de fonctions, de méthodes et de traitement.

Objectifs

- Distribution du codage dans une équipe
- Possibilité de faire des tests unitaires
- Réutilisation de codes
- Réalisation de bibliothèques logicielles

Objectifs de la modularité 2 / 2

Comment réaliser un programme modulaire ?

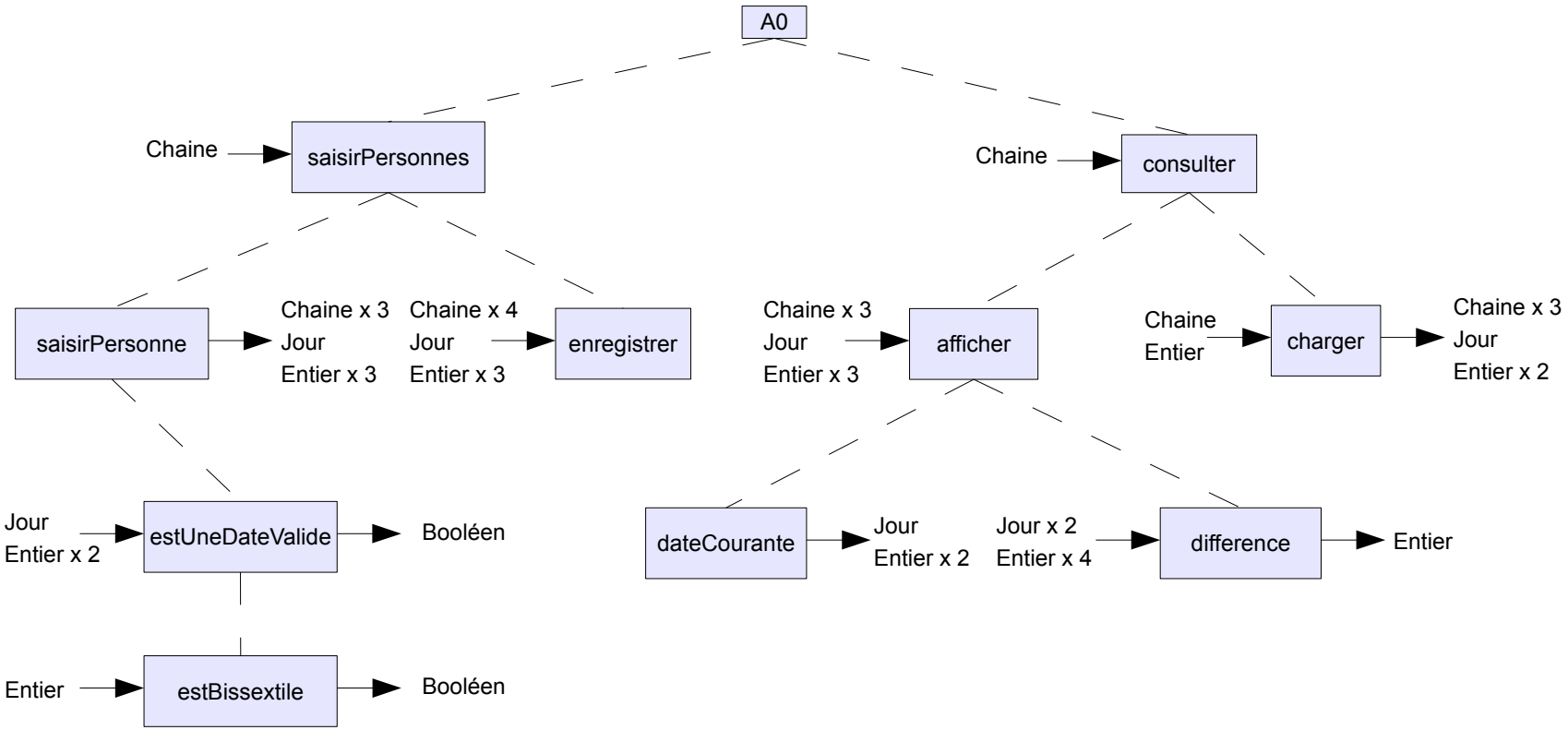
- Décomposition des traitements
- Découplage des sous-programmes
- Définition claire des interfaces
- Rédaction de documentation et de commentaires (symboles { } ou (* *) en Pascal)

Exemples d'analyse descendante 1 / 9

Carnet d'adresses

Faire un logiciel qui enregistre dans un fichier une liste de contacts indiquant leurs noms, prénoms, dates de naissance et adresses postales. Chaque personne est identifiée par un numéro unique. Le programme doit permettre de consulter ce carnet d'adresses et d'afficher le nombre de jours qui sépare la date courante de l'anniversaire d'une personne donnée.

Exemples d'analyse descendante 2 / 9



Exemples d'analyse descendante 3 / 9

Signatures des sous-programmes (1)

Type Jour = 1..31

{lance la saisie de plusieurs personnes et les enregistre dans le fichier nomFichier}
procédure saisirPersonnes **(E** nomFichier : **Chaine de caracteres)**

{Demande la saisie des numero, nom, prenom, adresse et date de naissance d'une personne, en s'assurant que la date est valide}
procédure saisirPersonne **(S** numero : **Entier**, nom, prenom, adresse : **Chaine de caracteres**, jour : Jour, mois, annee : **Entier)**

{renvoie vrai si les parametres d'entree correspondent à une date valide, faux sinon}
fonction estUneDateValide **(**jour : Jour, mois : **Entier**, annee : **Entier) : Booleen**

{renvoie vrai si l'année designée en paramètre d'entrée est bissextile, faux sinon}
fonction estBissextile **(**annee : **Entier) : Booleen**

{Enregistre dans le fichier de nom nomFichier une personne avec les valeurs des paramètres d'entrée}
procédure enregistrer **(E** nomFichier : **Chaine de caracteres**, numero : **Entier**, nom, prenom, adresse : **Chaine de caracteres**, jour : Jour, mois, annee : **Entier)**

Exemples d'analyse descendante 4 / 9

Signatures des sous-programmes (2)

{permet de consulter les contacts enregistrés dans le fichier nomFichier}
procédure consulter **(E** nomFichier : **Chaine de caracteres)**

{Recherche dans le fichier nomFichier la personne de numero num et affecte les valeurs correspondant à cette personne dans les paramètres de sortie. Si la personne n'existe pas seul le nom est modifié pour contenir une chaîne vide.}
procédure charger **(E** nomFichier : **Chaine de caracteres**, numero : **Entier**, **S** nom, prenom, adresse : **Chaine de caracteres**, jour : Jour, mois, annee : **Entier)**

{Affiche les informations liées à un contact et le nombre de jours avant son anniversaire.}
procédure afficher **(E** numero : **Entier**, nom, prenom, adresse : **Chaine de caracteres**, jour : Jour, mois, annee : **Entier)**

{renvoie le jour le mois et l'année actuelle}
procédure dateCourante **(S** jour : Jour, mois : **Entier**, annee : **Entier)**

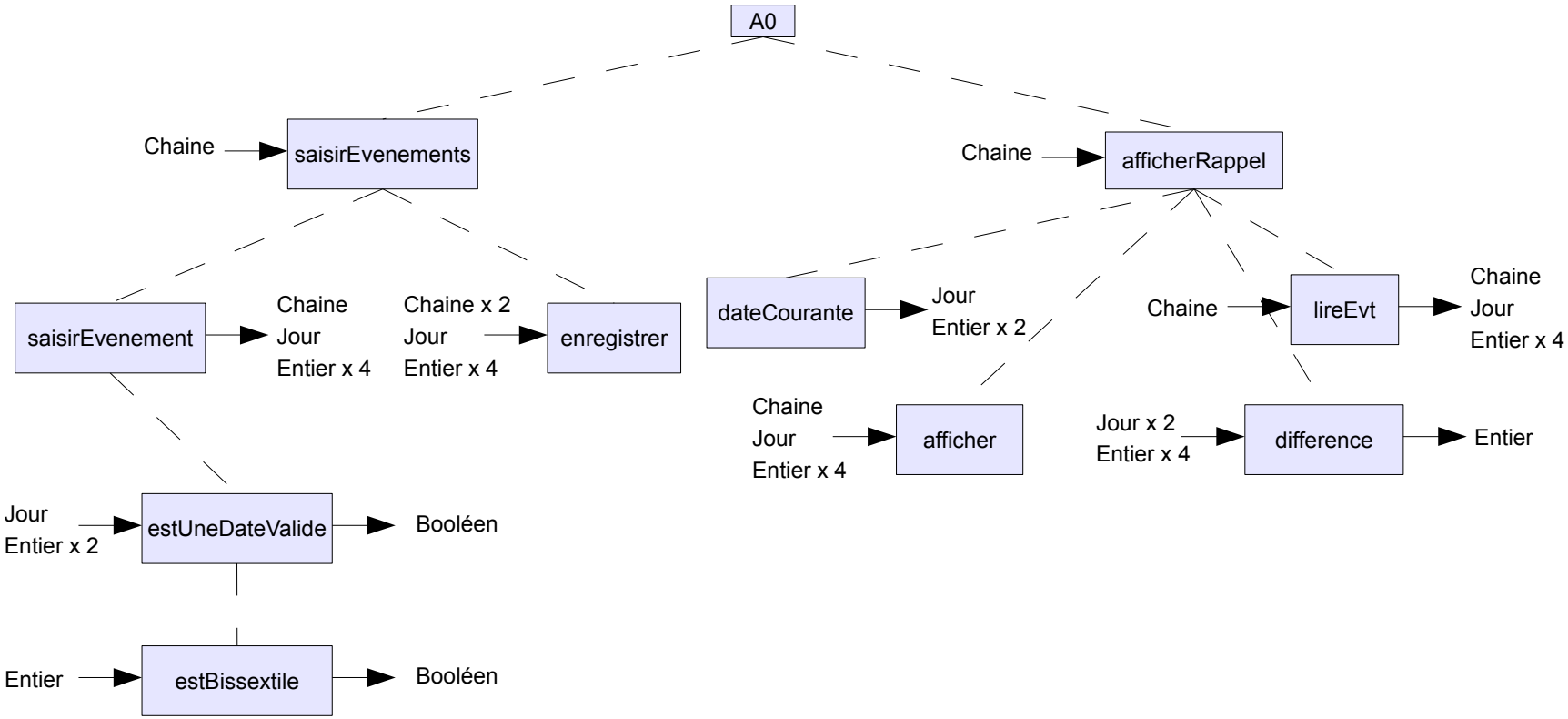
{renvoie le nombre de jours d'écart entre 2 dates}
fonction difference **(**jour1 : Jour, mois1 : **Entier**, annee1 : **Entier**, jour2 : Jour, mois2 : **Entier**, annee2 : **Entier) : Entier**

Exemples d'analyse descendante 5 / 9

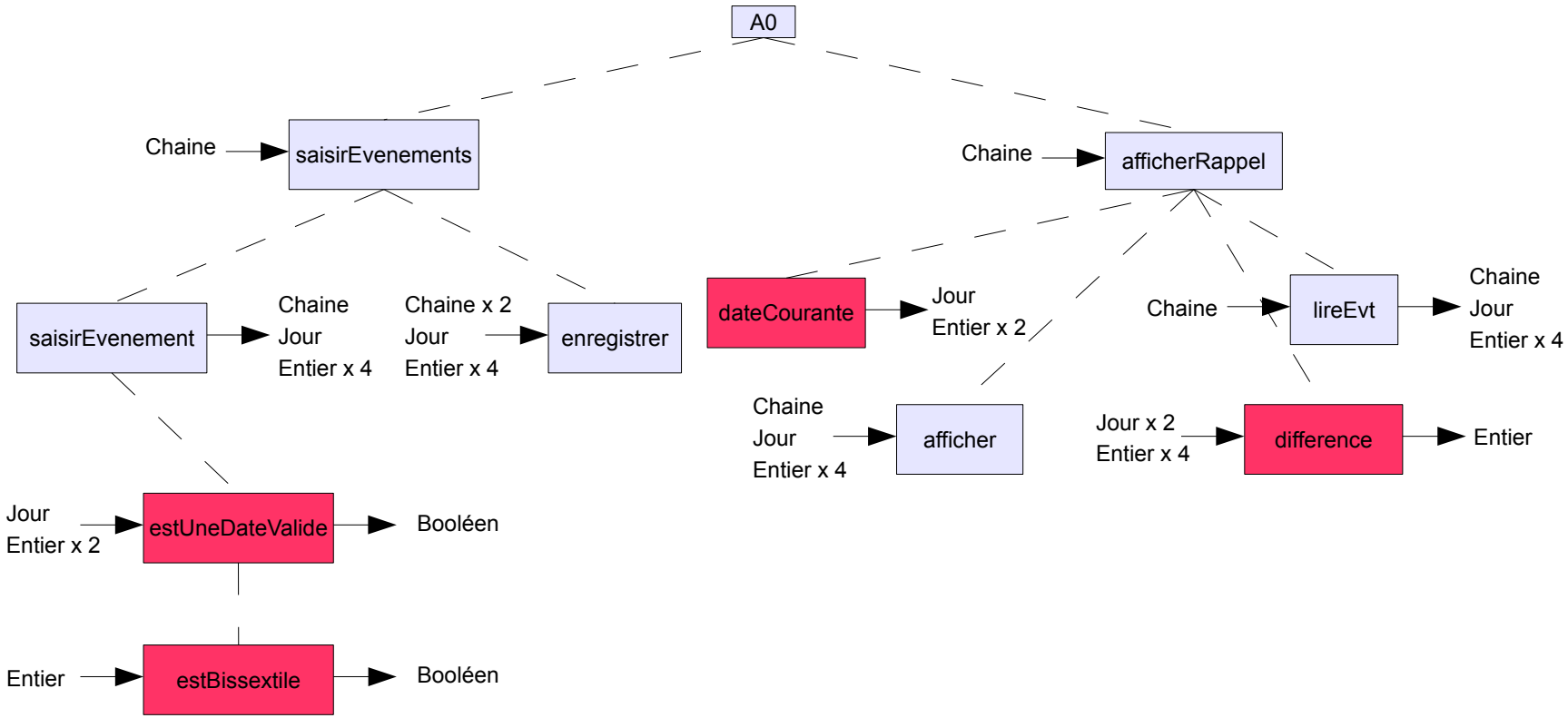
Emploi du temps

Faire un logiciel qui gère l'emploi du temps d'une personne. L'emploi du temps contient un ensemble d'événements. Chaque événement a un nom et se déroule à une date donnée entre 2 horaires. Au démarrage, le programme affiche un rappel de tous les événements qui se déroulent dans les 7 jours à venir.

Exemples d'analyse descendante 6 / 9



Exemples d'analyse descendante 7 / 9



Exemples d'analyse descendante 8 / 9

Module gestion de dates : signatures des sous-programmes

Type Jour = 1..31
{renvoie vrai si les paramètres d'entrée correspondent à une date valide, faux sinon}
fonction estUneDateValide **(**jour : Jour, mois : **Entier**, annee : **Entier) : Booleen**

{renvoie vrai si l'année designée en paramètre d'entrée est bissextile, faux sinon}
fonction estBissextile **(**annee : **Entier) : Booleen**

{renvoie le jour le mois et l'année actuelle}
procédure dateCourante **(S** jour : Jour, mois : **Entier**, annee : **Entier)**

{renvoie le nombre de jours d'écart entre 2 dates}
fonction difference **(**jour1 : Jour, mois1 : **Entier**, annee1 : **Entier**, jour2 : Jour, mois2 : **Entier**, annee2 : **Entier) : Entier**

Exemples d'analyse descendante 9 / 9

Signatures des sous-programmes spécifiques

{lance la saisie de plusieurs événements et les enregistre dans le fichier nomFichier}
procédure saisirEvenements **(E** nomFichier : **Chaine de caracteres)**

{Saisie des nom, date et heures d'un événement, en s'assurant que la date est valide}
procédure saisirEvenement **(S** nom : **Chaine de caracteres**, heureDebut, heureFin : **Entier**, jour : Jour, mois, annee : **Entier)**

{Enregistre dans le fichier de nom nomFichier un événement avec les valeurs des parametres d'entree}
procédure enregistrer **(E** nomFichier : **Chaine de caracteres**, nom : **Chaine de caracteres**, heureDebut, heureFin : **Entier**, jour : Jour, mois, annee : **Entier)**

{lit le fichier nomFichier et affiche tous les événements pour les 7 jours à venir.}
procédure afficherRappel **(E** nomFichier : **Chaine de caracteres)**

{Lit le fichier fic et affecte aux paramètres de sortie les valeurs correspondant au prochain événement dans le fichier. Le nom de l'événement est vide s'il n'y en a pas.}
procédure lireEvt **(E** fic : **Chaine de caracteres**, **S** nom : **Chaine de caracteres**, heureDebut, heureFin : **Entier**, jour : Jour, mois, annee : **Entier)**

{Affiche les informations liées à un événement.}
procédure afficher **(E** nom : **Chaine de caracteres**, heureDebut, heureFin : **Entier**, jour : Jour, mois, annee : **Entier)**

Modularité

- Séparer en fichiers sources différents
  - Réutilisables
    - par rôle / famille fonctionnelle
  - Spécifiques
- Chaque élément est testé
- Chaque élément est documenté
- L'intégration est contrôlée

Unités en Pascal 1 / 5

Définition générale

Le rôle d'une unité est de **regrouper** dans un même module l'implémentation d'un **ensemble de définitions de types, de fonctions et de procédures** d'un **même thème** dans un objectif de **réutilisabilité**.

- une unité est une bibliothèque logicielle,
- diffusée en dehors de l'équipe de programmation ou non
- nécessite une documentation

Unités en Pascal 2 / 5

Deux types d'unités :

On peut

- créer ses propres unités [génie logiciel]
- utiliser des unités existantes (bibliothèques)

Unités en Pascal 3 / 5

Deux types d'unités :

Créer ses propres unités [génie logiciel]

Syntaxe générale

```
unit <nom>;

interface

{declaration des types, constantes et signatures de
 sous-programmes utilisables par les programmes autres que
 l'unite}

implementation

{declaration des types, constantes, ... uniquement visibles dans
 l'unite}

{implementation des sous-programmes}
end.
```

Unités en Pascal 4 / 5

dates.pas

```
unit dates;

interface

Type Jour = 1..31;

function estBissextile (annee : integer): boolean;
function estUneDateValide(jour:Jour;mois,annee:integer):boolean;

implementation

{renvoie vrai si l'annee designee en parametre d'entree
est bissextile, faux sinon}
function estBissextile (annee : integer): boolean;
begin
   estBissextile := ((annee mod 400) = 0) or
   (((annee mod 4) = 0) and ((annee mod 100) <> 0));
end; { estBissextile }
```

Unités en Pascal 5 / 5

dates.pas

```
{renvoie vrai si les parametres d'entree correspondent a
une date valide, faux sinon}
function estUneDateValide(jour:Jour;mois,annee:integer):boolean;
var valide :  boolean;
begin
   valide := true;
   case mois of
     1,3,5,7,8,10,12 : valide := true;
     4,6,9,11        :
     if (jour <> 31) then valide := true;
     2 :
         if (jour < 29) then valide := true
         else if (estBissextile(annee)) and (jour = 29) then
            valide := true
         else
            valide :=false;
   end; { case }
   estUneDateValide := valide;
end; { estUneDateValide }

end.
```

Usage des unités

Syntaxe

pour utiliser le contenu d'une unité dans un autre programme, il faut l'"importer" par la commande `uses`.

datesMain.pas

```
program bissextile;

uses dates;

var i : integer;
begin
   for i := 1900 to 2013 do
      if (estBissextile(i)) then
         writeln(i, ' est bissextile');
end.
```

Usage des unités

Syntaxe

pour utiliser le contenu d'une unité dans un autre programme, il faut l'"importer" par la commande `uses`.

Interface

Tous les éléments déclarés dans l'interface sont utilisables :

- Constantes, types personnalisés
- Fonctions, procédures

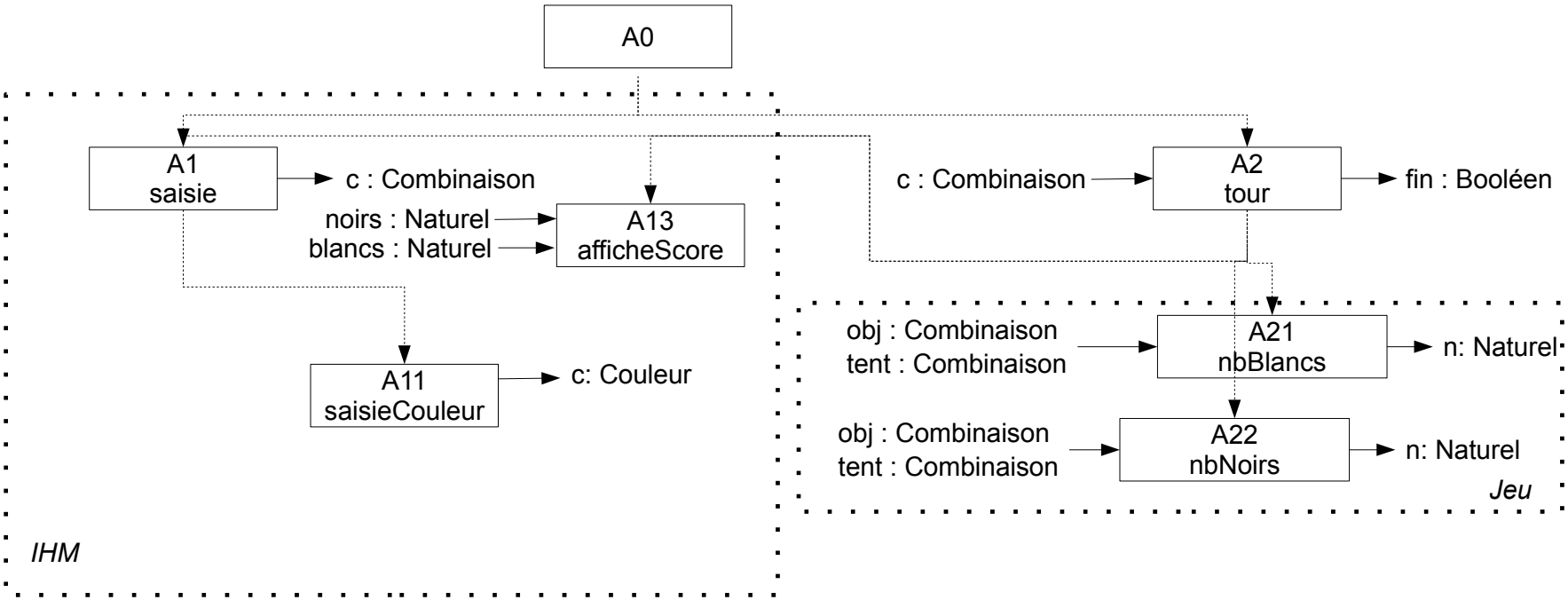
Exemple :Jeu du mastermind 1 / 2

Quelques fonctions et procédures

Écrire une nouvelle version du jeu du mastermind. Cette version s'appuiera sur des unités pour :

1. séparer le code source en une unité pour la gestion de l'interface homme-machine, une pour le calcul des pions blancs et noirs des règles du jeu et une pour le programme principal gérant l'ensemble du jeu ;

Exemple :Jeu du mastermind 2 / 2



Compilation et édition de liens

Compilation

Le processus de compilation créé un fichier objet à partir d'un fichier source.
Par exemple `fpc dates.pas` provoque la création de `dates.o` (et `dates.ppu`) si la compilation réussit.

Édition de liens

Le processus d'éditions de liens lie plusieurs fichiers objets pour créer un programme exécutable. Le compilateur `fpc` effectue l'édition de lien s'il est appelé sur un fichier source contenant un programme principal.

Utilisation

Deux types d'unités :

On peut

- créer ses propres unités [génie logiciel]
- utiliser des unités existantes (bibliothèques) :
  `https://www.freepascal.org/docs-html/rtl/index.html`
  - Unité system importée systématiquement
  - e.g. readln, writeln...
    `https://www.freepascal.org/docs-html/rtl/system/index.html`

documentation

Problème

Question

∃ une fonction round(r : Real)

- quel est son type de retour ?
- quelle est sa valeur de retour ?
  - 8.5, est-ce 8 ou 9 ?
  - 3.5, est-ce 3 ou 4 ?

Problème

Question

∃ une fonction round(r : Real)

- quel est son type de retour ?
- quelle est sa valeur de retour ?
  - 8.5, est-ce 8 ou 9 ?
  - 3.5, est-ce 3 ou 4 ?

`https://www.freepascal.org/docs-html/rtl/index.html`

Utilisation

- Pour savoir ce qui existe, consulter la documentation
  `https://www.freepascal.org/docs-html/rtl/index.html`
- Utiliser votre moteur de recherche favori pour retrouver les éléments
  `https://www.qwant.com/?q=freepascal%20convert%20string%20to%20integer&t=web`

Documentation : pasdoc

Pour maintenir / partager son code

- Générateurs de documentation (doxygen, javadoc...)
- Pascal : pasdoc

Principes

- Extraction de documentation depuis les commentaires
- Utilisation de balises (e.g. @author, @includeCode...)
- Exemple : `code/parfaitUtils.html`

Source

Nombres parfaits

```
{@author(Julien Saunier)}

{ Elements de calcul des nombres parfaits}
unit parfaitUtils;
interface
{valeur maximale testee}
const MAX_PARFAIT = 32766;

{ calcule si a est un diviseur de b}
function estUnDiviseur(a,b : Integer) : Boolean;
{ calcule la somme des diviseurs de n}
function sommeDesDiviseurs(n :  Integer) : Integer;
{ verifie si n est un nombre parfait, i.e. egal a la somme de ses diviseurs propres

@longCode( writeln(estParfait(8128)) ) affiche True}
function estParfait(n : Integer) : Boolean;

{saisie d'une borne par un utilisateur }
function obtenirBorneMax() : Integer;

{procedure affichant les nombres parfait jusqu'a une borne n}
procedure afficherNombresParfaitsJusquA(n : Integer);

procedure afficherDesNombresParfaits();
```

Génération de documentation

Génération de la documentation

Dans le terminal, utiliser (dans le répertoire du fichier source) :

```
pasdoc source.pas
```

Génère source.html / pasdoc.css

Tests (unitaires)

Notion de test

Définition (wikipédia)

En programmation informatique, le test unitaire [...] est une procédure permettant de vérifier le bon fonctionnement d'une partie précise d'un logiciel ou d'une portion d'un programme [...].

- temps de débogage d'un programme $>$ son temps d'écriture
- d'autant plus long que complexe (enchainement d'appels de sous-programmes)
- $\rightarrow$ tester chaque sous-élément

Tests unitaires

- tests automatisés (fpcunit, JUnit, ...) ou manuels (via un programme principal)
- utilisation de cas d'usages **représentatifs** pour vérifier que le sous-programme fonctionne correctement

Fibonacci

```
var j : Integer;
begin
        for j := 1 to 100 do
                writeln(j, ' ', fibo(j))

end.
```

ou...

```
if fibo (1) <> 1 then
        writeln ('erreur: 1');
if fibo(7) <> 21 then
        writeln ('erreur: 7');
```

Tests unitaires & unités

- Unités = pas de programme principal
- $\Rightarrow$ faire un programme principal temporaire pour tester les fonctions et procédures au fur et à mesure
- Possibilité d'externaliser en passant des paramètres au programme par la ligne de commande

Paramètres du programme principal 1 / 2

- Deux fonctions
  - paramCount() : Nombre de paramètres
  - paramStr(n) : retourne la chaîne de caractères en n^{ieme} position
    paramStr(0) est le nom du programme

Programme principal

```
program mainProg;

var i : Integer;
begin
        for i := 0 to paramCount do
                writeln(i, ' ', paramStr(i))
end.
```

Paramètres du programme principal 2 / 2

Exemple nombres parfaits

```
program nbparfait;
uses parfaitUtils, sysutils;

procedure afficherDesNombresParfaitsParam();
var nb :  Integer;
begin
   if paramCount() > 0 then
        begin
                nb:=StrToInt(paramStr(1));
                afficherNombresParfaitsJusquA(nb)
        end
   else
                writeln('donner le nombre en parametre!');
end;

begin
   afficherDesNombresParfaitsParam()
end.
```