```
int fact(int n);              // Declaration de la fonction 'fact()'

int fact(int n)               // Definition de la fonction 'fact()'
{
if(n>1) return(n*fact(n-1)); // Appel recursif
else return(1);
}

void afficher_fact(int n)     // Definition de la fonction 'afficher_fact()'
{
println("fact(",n,") = ",fact(n));
}

execute                       // Code a executer
{
println("---- debut ----");
int i;                        // Declaration d'un entier 'i'
for(i=1;i<=5;i++)
  {
  afficher_fact(i);           // Utiliser la fonction 'afficher_fact()'
  }
println("---- fin ----");
}
```

```
---- debut ----
fact(1) = 1
fact(2) = 2
fact(3) = 6
fact(4) = 24
fact(5) = 120
---- fin ----
```

Figure 1 : Des fonctions en *oRis*

```
class Exemple               // Declaration de la classe 'Exemple'
{
int i;                      // Un 'Exemple' contient un entier 'i'
void new(void);             // Un 'Exemple' dispose d'un constructeur
void delete(void);          // Un 'Exemple' dispose d'un destructeur
void show(void);            // Un 'Exemple' dispose d'une methode 'show()'
};

void Exemple::new(void)     // Definition du constructeur
{                           // de la classe 'Exemple'
println(this," apparait");
}

void Exemple::delete(void)  // Definition du destructeur
{                           // de la classe 'Exemple'
println(this," disparait");
}

void Exemple::show(void)    // Definition de la methode 'show()'
{                           // de la classe 'Exemple'
println(this," : i = ",i);
}

execute                     // Code a executer
{
println("---- debut ----");
Exemple t1=new Exemple;     // Instancier deux 'Exemple'
Exemple t2=new Exemple;
t1->i=5;                    // Modifier leur attribut 'i'
t2->i=t1->i+3;
t1->show(); t2->show();     // Utiliser leur methode 'show()'
delete t1; delete t2;       // Detruire les deux 'Exemple'
println("---- fin ----");
}
```

```
---- debut ----
Exemple.1 apparait
Exemple.2 apparait
Exemple.1 : i = 5
Exemple.2 : i = 8
Exemple.1 disparait
Exemple.2 disparait
---- fin ----
```

Figure 2 : Des objets en *oRis*

```
class Exemple                 // Declaration de la classe 'Exemple'
{
int age;                      // Un 'Exemple' contient un entier 'age'
void new(void);               // Un 'Exemple' dispose d'un constructeur
void delete(void);            // Un 'Exemple' dispose d'un destructeur
void main(void);              // Un 'Exemple' dispose d'une methode 'main()'
};                            //                   ---> c'est un objet actif

void Exemple::new(void)       // Definition du constructeur
{age=0;}                      // de la classe 'Exemple'

void Exemple::delete(void)    // Definition du destructeur
{}                            // de la classe 'Exemple'

void Exemple::main(void)      // Definition de la methode 'main()'
{                             // de la classe 'Exemple'
println("L'age de ",this,
        " est ",++age);
}

execute                       // Code a executer (initialisation)
{
println("---- debut ----");
for(int i=0;i<5;i++)          // Instancier cinq 'Exemple'
  new Exemple;                // non transitoires grace a leur 'main()'
println("---- fin ----");
}
```

```
---- debut ----
---- fin ----
L'age de Exemple.2 est 1
L'age de Exemple.5 est 1
L'age de Exemple.3 est 1
L'age de Exemple.1 est 1
L'age de Exemple.4 est 1
L'age de Exemple.1 est 2
L'age de Exemple.3 est 2
L'age de Exemple.4 est 2
L'age de Exemple.2 est 2
L'age de Exemple.5 est 2
...
```

Figure 3 : Des objets actifs en *oRis*

```
class Info : Message                // Un nouveau type de 'Message'
{
string texte;
void new(string msg);
void delete(void);
};

void Info::new(string msg)
{
texte=msg;
}

void Info::delete(void) {}

class Exemple : Agent               // Les 'Agent' savent utiliser la
{                                   // communication point a point
void new(void);
void delete(void);
void main(void);
};
void Exemple::new(void) {}
void Exemple::delete(void) {}

void Exemple::main(void)            // Consultation des messages
{
Info info=new Info(?(2) ? "ceci" : "cela");
Exemple[] l=(Exemple[])getAll("Exemple");
info->sendTo(l[?(l.length())]);     // Envoyer un message
println(this," lecture :");         // Lire les messages
while(getNbMessages())
  {
  info=(Info)getNextMessage();
  println("  de ",info->getEmitter()," : ",info->texte);
  }
}

execute                             // Code a executer (initialisation)
{
println("---- debut ----");
for(int i=0;i<3;i++)                // Instancier trois 'Exemple'
  NEW Exemple;                      // non transitoires grace a leur 'main()'
println("---- fin ----");
}
```

Figure 4 : Communication point à point en *oRis* (1/2)

```
---- debut ----
---- fin ----
Exemple.3 lecture :
Exemple.1 lecture :
  de Exemple.3 : ceci
Exemple.2 lecture :
Exemple.3 lecture :
  de Exemple.1 : ceci
  de Exemple.2 : cela
Exemple.1 lecture :
  de Exemple.3 : cela
Exemple.2 lecture :
Exemple.2 lecture :
Exemple.3 lecture :
  de Exemple.1 : ceci
Exemple.1 lecture :
  de Exemple.2 : ceci
  de Exemple.2 : cela
...
```

Figure 5 : Communication point à point en *oRis* (2/2)

```
class Alerte : Message                            // Un nouveau type de 'Message'
{
void new(void);
void delete(void);
};
void Alerte::new(void) {}
void Alerte::delete(void) {}

class Humain : Agent                              // Les 'Agent' savent utiliser
{                                                 // la diffusion
void new(void);
void delete(void);
void fuir(Alerte alerte);
};
void Humain::new(void)                            // La diffusion d'une 'Alerte'
{setSensitivity("Alerte","fuir");}                // doit declencher 'fuir()'
void Humain::delete(void) {}
void Humain::fuir(Alerte alerte)
{println(this,"->FUIR(",alerte,")");}

class Incendie : Alerte                           // Une 'Alerte' particuliere
{
void new(void);
void delete(void);
};
void Incendie::new(void) {}
void Incendie::delete(void) {}

class Pompier : Humain                            // Un 'Humain' particulier
{
void new(void);
void delete(void);
void eteindre(Incendie incendie);
};
void Pompier::new(void)                           // La diffusion d'un 'Incendie'
{setSensitivity("Incendie","eteindre");}          // doit declencher 'eteindre()'
void Pompier::delete(void) {}
void Pompier::eteindre(Incendie incendie)
{println(this,"->ETEINDRE(",incendie,")");}
```

Figure 6 : Diffusion de messages en *oRis* (1/2)

```
class Inondation : Alerte                         // Une 'Alerte' particuliere
{
void new(void);
void delete(void);
};
void Inondation::new(void) {}
void Inondation::delete(void) {}

class Plombier : Humain                           // Un 'Humain' particulier
{
void new(void);
void delete(void);
void colmater(Inondation inondation);
};
void Plombier::new(void)
{setSensitivity("Inondation","colmater");}
void Plombier::delete(void) {}
void Plombier::colmater(Inondation inondation) // La diffusion de 'Inondation'
{println(this,"->COLMATER(",inondation,")");}  // doit declencher 'colmater()'

execute                                           // Code a executer
{
println("---- debut ----");
NEW Humain; NEW Pompier; NEW Plombier;            // Creer des agents
new Alerte->broadcast();                          // Diffuser des messages
new Incendie->broadcast();
new Inondation->broadcast();
println("---- fin ----");
}
```

```
---- debut ----
Plombier.1->FUIR(Alerte.1)
Pompier.1->FUIR(Alerte.1)
Humain.1->FUIR(Alerte.1)
Humain.1->FUIR(Incendie.1)
Pompier.1->ETEINDRE(Incendie.1)
Plombier.1->FUIR(Incendie.1)
Humain.1->FUIR(Inondation.1)
Pompier.1->FUIR(Inondation.1)
Plombier.1->COLMATER(Inondation.1)
---- fin ----
```

Figure 7 : Diffusion de messages en *oRis* (2/2)

```
//===================================== Par l'intermediaire d'une IHM ========
...                   //---- Saisie du code en direct
execute
{ ... }

include "newCode.ors" //---- Chargement d'un fichier deja redige

//========================================== Dans du code deja charge ========
{                     //---- Code introduisant explicitement du nouveau code
parse(" ... code oRis ... ");
...
}

{                     //---- Code chargeant du code deja redige
load("newCode.ors");
...
}
```

Figure 8 : Introduction de nouveau code en *oRis*

```
void exemple(void)                         // Definition de la fonction
{                                          // 'exemple()'
println("debut exemple initial");
parse("void exemple(void)
       {println(\"nouvel exemple\");}"); // Surdefinition de 'exemple()'
println("fin exemple initial");
}

execute                                    // Code a executer
{
println("---- debut ----");
exemple();                                 // Premier appel a 'exemple()'
exemple();                                 // Deuxieme appel a 'exemple()'
println("---- fin ----");
}
```

```
---- debut ----
debut exemple initial
fin exemple initial
nouvel exemple
---- fin ----
```

Figure 9 : Surdéfinition d'une fonction

```
execute                                   // Code a executer
{
println("---- debut ----");
string nouveauNom="Exemple";              // La nouvelle classe sera 'Exemple'
string program="
  class " + nouveauNom + "                // Declarer la nouvelle classe
  {
  void new(void);
  void delete(void);
  void main(void);
  };
  void " + nouveauNom + "::new(void) {}  // Definir son code
  void " + nouveauNom + "::delete(void) {}
  void " + nouveauNom + "::main(void)
  {println(this,\" est un " + nouveauNom + " !\");}
  execute {new " + nouveauNom + ";}      // En creer une instance
  ";
println(program);                         // Afficher la chaine
parse(program);                           // Interpreter la chaine composee
println("---- fin ----");
}
```

```
---- debut ----

  class Exemple              // Declarer la nouvelle classe
  {
  void new(void);
  void delete(void);
  void main(void);
  };
  void Exemple::new(void) {}  // Definir son code
  void Exemple::delete(void) {}
  void Exemple::main(void)
  {println(this," est un Exemple !");}
  execute {new Exemple;}      // En creer une instance

---- fin ----
Exemple.1 est un Exemple !
Exemple.1 est un Exemple !
Exemple.1 est un Exemple !
...
```

Figure 10 : Déclaration dynamique d'une nouvelle classe

```
//=============================================== Concernant l'heritage ========

class Exemple;              // Declaration d'une classe 'Exemple'
                            // elle herite implicitement de 'Object'

class Exemple : Agent;      // Elle herite maintenant de 'Agent'

class Exemple : A,B;        // Elle herite en plus de 'A' et 'B'

//=============================================== Concernant les methodes ======

void Exemple::f(void);      // Ajout et declaration de 'f()'

void Exemple::f(void) {...} // Definition de 'f()'

void Exemple::g(void) {...} // Ajout et definition de 'g()'

void Exemple::g(void) {...} // Surdefinition de 'g()'

void Exemple::g(void);      // 'g()' n'est plus definie (seulement declaree)

//=============================================== Concernant les attributs =====

int Exemple::i;             // Ajout d'un attribut entier 'i'

Object[] Exemple::l;        // Ajout d'un attribut liste d'objets 'l'


//============================================================================//
//                                                                            //
// Les modifications sont toujours des ajouts ou des surdefinitions          //
//                                                                            //
// Les surdefinitions doivent respecter les prototypes (pas de surcharge)    //
//                                                                            //
// Il est impossible :                                                        //
//    - d'annuler un heritage,                                                //
//    - de supprimer une methode,                                             //
//    - de supprimer un attribut.                                             //
//                                                                            //
//============================================================================//
```

Figure 11 : Modification d'une classe existante

```
class Exemple                        // Declaration de la classe 'Exemple'
{
void new(void);
void delete(void);
void main(void);
};

void Exemple::new(void) {}
void Exemple::delete(void) {}
void Exemple::main(void)             // Methode 'main()' initiale
{println(this," est un Exemple !");}

execute                              // Code a executer
{
println("---- debut ----");
for(int i=0;i<3;i++) new Exemple;   // Instancier trois 'Exemple'
string program="
  int Exemple.2::compteur;           // Ajouter un attribut a 'Exemple.2'
  void Exemple.2::afficher(void)     // Ajouter une methode a 'Exemple.2'
  {
  println(this,\" compte \",compteur);
  }
  void Exemple.2::main(void)         // Surdefinir la methode 'main()' de
  {                                  // 'Exemple.2' afin d'utiliser
  compteur++;                        // ce qui a ete ajoute
  afficher();
  }";
parse(program);                      // Interpreter la chaine composee
println("---- fin ----");
}
```

```
---- debut ----
---- fin ----
Exemple.1 est un Exemple !
Exemple.3 est un Exemple !
Exemple.2 est un Exemple !
Exemple.1 est un Exemple !
Exemple.3 est un Exemple !
Exemple.2 compte 1
Exemple.1 est un Exemple !
Exemple.2 compte 2
Exemple.3 est un Exemple !
...
```

Figure 12 : Modification du comportement d'une instance

```
include "attrCB.pkg"              // Necessaire pour les liens reflexes

class Exemple                     // Declaration d'une classe 'Exemple'
{
int valeur;
void new(int valeurInit);
void delete(void);
};
void Exemple::new(int valeurInit)
{valeur=valeurInit;}
void Exemple::delete(void) {}

class Lien : AttrCB               // Declaration d'une classe 'Lien'
{                                 // Heritant de 'AttrCB'
Exemple _src,_dest;
int _mem;
void new(Exemple src,Exemple dest);
void delete(void);
void before(void);
void after(void);
};

void Lien::new(Exemple src,Exemple dest)
{
AttrCB::new(src,"valeur");        // Se connecter a l'attribut de la source
_src=src;                         // Memoriser le nom de la source
_dest=dest;                       // Memoriser le nom de la destination
}

void Lien::delete(void) {}

void Lien::before(void)
{
println(this," : debut de before()");
_mem=_src->valeur;                // Memoriser l'ancienne valeur source
}

void Lien::after(void)
{
_dest->valeur+=_src->valeur-_mem; // Propager la variation
println(this," : fin de after()");
}
```

Figure 13 : Utilisation de liens réflexes (1/2)

```
execute                                                  // Code a executer
{
println("---- debut ----");
Exemple ex1=new Exemple(10);                             // Instancier et initialiser
Exemple ex2=new Exemple(20);                             // trois 'Exemple'
Exemple ex3=new Exemple(30);
Lien l1=new Lien(ex1,ex2);                               // Les relier en cycle
Lien l2=new Lien(ex2,ex3);                               // par trois 'Lien'
Lien l3=new Lien(ex3,ex1);
ex1->valeur=12;                                          // Modifier un 'Exemple'
println(ex1->valeur,"\t",ex2->valeur,"\t",ex3->valeur);
ex2->valeur=25;                                          // En modifier un autre
println(ex1->valeur,"\t",ex2->valeur,"\t",ex3->valeur);
ex3->valeur=48;                                          // En modifier un autre
println(ex1->valeur,"\t",ex2->valeur,"\t",ex3->valeur);
println("---- fin ----");
}
```

```
---- debut ----
Lien.1 : debut de before()
Lien.2 : debut de before()
Lien.3 : debut de before()
Lien.3 : fin de after()
Lien.2 : fin de after()
Lien.1 : fin de after()
12      22      32
Lien.2 : debut de before()
Lien.3 : debut de before()
Lien.1 : debut de before()
Lien.1 : fin de after()
Lien.3 : fin de after()
Lien.2 : fin de after()
15      25      35
Lien.3 : debut de before()
Lien.1 : debut de before()
Lien.2 : debut de before()
Lien.2 : fin de after()
Lien.1 : fin de after()
Lien.3 : fin de after()
28      38      48
---- fin ----
```

Figure 14 : Utilisation de liens réflexes (2/2)

```
//-------------------------------------------------- Type logique # type entier
replace bool by int;
replace true by 1;
replace false by 0;

class Object //------------------ Classe de base de toutes les autres classes
{                                                 // Contruction / destruction
void   new(void);
void   delete(void);
void   deleteOnExit(void);
                                                  // Identification
string name(void);
bool   isA(string className);
string className(void);
int    suffix(void);

bool   equalsTo(Object obj);                      // Operateurs
bool   lessThan(Object obj);
void   affect(Object obj);
                                                  // Controle des objets actifs
void   suspend(void);
void   resume(void);
bool   isSuspended(void);
                                                  // Ramasse-miettes
void   setTransient(bool isTransient);
bool   isTransient(void);
};

class Agent //---------------- Classe permettant une communication ‘‘evoluee’’
{                                                 // Contruction / destruction
void    new(void);
void    delete(void);
                                                  // Diffusion
void    getSensitivity(string[] & msgTypes,string[] & callbacks);
void    setSensitivity(string msgType,string callback);
void    cancelSensitivity(string msgType);
void    asynchronousReception(Message message);
                                                  // Point a point
int     getNbMessages(void);
Message peekMessage(int pos);
Message getNextMessage(void);
void    putBackMessage(Message message);
void    onMessage(void);
};
```

Figure 15 : Les notions prédéfinies en *oRis* (1/4)

```
class Message //----------- Support de base pour une communication ‘‘evoluee’’
{                                                 // Contruction / destruction
void   new(void);
void   delete(void);
                                                  // Diffusion
void   broadcast(void);
                                                  // Point a point
void   sendTo(Agent receiver);
Object getEmitter(void);
void   setEmitter(Object emitter);
};

//------------------------------------------------------------------ Fonctions
                                                  // Controle de l’execution
replace  randomPicking by 0;
replace  serialPicking by 1;
int      getPickingMode(void);
void     setPickingMode(int mode);
int      getInterval(void);
void     setInterval(int interval);
void     yield(void);
void     softYield(void);
void     getExecutionStack(Object[] & callers,string[] & modules,
                           int execution);
void     printStack(void);
int      currentExecution(void);
void     abort(int execution);
void     suspend(int execution);
void     resume(int execution);
bool     isSuspended(int execution);
string   control(string command,string argument);
void     onExit(string command);
                                                  // Controle des recursions
int      getMaxDepth(void);
void     setMaxDepth(int maxDepth);
                                                  // Introspections simples
Object   getObject(string objName);
bool     exist(Object obj);
int      numberOf(string className);
Object[] getAll(string className);
string[] getClasses(void);
                                                  // Controle du temps
int      getClock(void);
int      getTime(void);
```

Figure 16 : Les notions prédéfinies en *oRis* (2/4)

```
                                                        // Utilitaires
int       system(string command);
void      systemI(string command,string input);
string    systemO(string command);
string    systemE(string command);
string    systemIO(string command,string input);
string    systemIE(string command,string input);
string    systemOE(string command,string & err);
string    systemIOE(string command,string input,string & err);
void      systemYield(void);
string    getenv(string varName);
void      putenv(string varName,string value);
string    systemName(bool detailed);
string[]  commandLine(bool full);
int       asciiCode(string char);
string    asciiChar(int code);
string    hexaString(int value);
string    octalString(int value);
string    binaryString(int value);
string    locateFile(string fileName);
void      showURL(string url,bool newWindow);
                                                        // Entrees/sorties de texte
void      print(type1 value1,type2 value2,...);
void      println(type1 value1,type2 value2,...);
string    format(type1 value1,type2 value2,...);
string    readln(void);
void      error(type1 value1,type2 value2,...);
                                                        // Introduction de code
bool      loadLibrary(string libName);
int       parse(string program);
int       load(string fileName);
int       eval(string code);
string[]  parseEval(string command);
```

Figure 17 : Les notions prédéfinies en *oRis* (3/4)

```
//---------------------------------------- Quelques mots clefs et expressions
                                                 // Manipulations de listes
data=l[pos];
l[pos]=data;
data=l.front();
l.front()=data;
data=l.back();
l.back()=data;
l.insert(pos,data);
l.pushFront(data);
l.pushBack(data);
data=l.remove(pos);
data=l.popFront();
data=l.popBack();
l.erase();
nb=l.length();
pos=l.seekMin();
pos=l.seekMax();
pos=l.seek(data);
pos=l.deepSeek(data);
l.sort();
                                                 // Manipulations de chaines
s2=s1[pos];
nb=s.length();
pos=s1.seek(s2);
// + package "convert.pkg"
                                                 // Manipulations "generiques"
string valueDump(type value);
void   valueParse(type & result,string constant);
// + package "contain.pkg"
```

Figure 18 : Les notions prédéfinies en *oRis* (4/4)