

# Production de code intermédiaire :

## Documentation partielle

---

Ce document a pour but de définir précisément le code C++ approprié à chaque élément du langage *WHILE*. Tous les exemples décrits ci-après se baseront sur des objets de type Arbre Binaire (BinTree). Toutes les variables nouvellement déclarées en CPP se voient attribuer la valeur *nullptr*.

## Traduction d'une fonction

---

Contenu	WHILE	CPP
Définition	function <b>name</b> :	List< BinTree > <b>name</b> (List< BinTree > args) {}
Paramètre(s)	<b>read VAR (,</b> <b>VAR2, ...,</b> <b>VARn)</b>	BinTree <b>VAR</b> = args.at <a href="http://args.at(x)">http://args.at(x)</a>
Retour(s)	<b>write VAR (,</b> <b>VAR2, ...,</b> <b>VARn)</b>	La fonction retourne une List< BinTree > contenant les variables retournées.

Exemple de traduction d'une fonction While en CPP :

```
function foo:  
read X  
%  
nop ;  
%
```

write Y

```
List<BinTree> foo(List<BinTree> args)
{
  BinTree X = args.at(0);
  List<BinTree> ret;
  //NOP
  BinTree Y;
  ret.add(Y);
  return ret;
}
```

## Traduction des instructions

Contenu	WHILE	CPP
Lecture(s)	read VAR (, VAR2, ..., VARn)	variable(s) locale(s) à déclarer
Affectation(s)	VAR(, VAR2, ..., VARn) := <b>something</b>	variable locale à déclarer (si inexistante) ET à affecter à la valeur something
Ne rien faire	nop	//NOP //NOP

## Traduction du **something**

WHILE	CPP
-------	-----

WHILE	CPP
nil	BinTree::NIL
VAR	VAR en la déclarant si pas déjà fait
(cons a b)	BinTree::cons({a,b})
(cons a b c d)	BinTree::cons({a,b,c,d})
(hd a)	BinTree::hd(a)
(tl a)	BinTree::tl(a)
(func a, b, c, etc.)	func({a,b,c,etc})

Exemple de traduction d'instructions While en CPP :

```
read X,Y
X:=cons(1 nil);
Y:=tl(X);
W:=hd(cons(1,cons(nil,nil)));
Z:=X;
nop;
```

```
BinTree X = args.at(0), Y = args.at(1);
X=BinTree::cons(1,BinTree::nil);
Y=BinTree::tl(X);
BinTree W = BinTree::hd(BinTree::cons(1,BinTree::cons(BinTree::nil,BinTree::nil)));
BinTree Z = X;
//NOP
```

# Traduction en code 3 adresses

## Tableau de correspondance

While	Code 3 adresses
X = nil	< nil, X, _, _ >
nop	< nop, _, _, _ >
X = (cons Y Z)	< cons, X, Y Z, _ >
X = (cons A B C D)	< cons, X, A B C D, _ >
X = (hd Y)	< hd, X, Y, _ >
X = (tl Y)	< tl, X, Y, _ >
X = Y =? Z	< =?, X, Y, Z >
X := Y	< :=, X, Y, _ >
if cond then codeThen else codeElse	if cond goTo then\n else: codeElse; goTo fi\n then: codeThen; fi
while cond then code od	while: if cond goTo then\n else: goTo fi\n then: code; goTo while; fi
for cond then code od	for: if cond goTo then\n else: goTo fi\n then: code; goTo for; fi

While	Code 3 adresses
foreach elem in ensemb do cmds od	foreach: if e:Var goTo then\n else: goTo fi\n then: code; goTo foreach; fi

Une variable créée par le compilateur aura la notation %vXX avec XX le numéro de variable.

## Exemples WHILE->3@

### Exemple 1

```
function foo:
read X, Z
%
nop ;
%
write Y
```

<nop, \_, \_, \_>

+ Table des symboles

### Exemple 2

```
function foo:
read X,Y
%
```

```
X:=cons(nil nil);  
Z:=X;  
nop;  
%  
write Z
```

```
<:=, %v1, nil, _>  
<:=, %v2, nil, _>  
<cons, %v3, %v1 %v2, _>  
<:=, X, %v3, _>  
<:=, Z, X, _>  
<nop, _, _, _>
```

## Exemple 3

```
function foo:  
read X,Y  
%  
X:=(cons nil nil);  
Y:=(tl X);  
W:=(hd (cons nil (cons nil nil)));  
Z:=X;  
nop;  
%  
write Z
```

```
<:=, %v1, nil, _>  
<:=, %v2, nil, _>  
<cons, %v3, %v1 %v2, _>  
<:=, X, %v3, _>
```

```
<tl, %v4, X, _>
<:=, Y, %v4, _>
<:=, %v5, nil, _>
<:=, %v6, nil, _>
<cons, %v7, %v5 %v6, _>
<:=, %v8, nil, _>
<cons, %v9, %v8 %v7, _>
<hd, %v10, %v9, _>
<:=, W, %v10>
<:=, Z, X, _>
<nop, _, _, _>
```

## Exemple 4

```
function foo:
  read X,Y
  %
  while X do
    if Y then
      Y:=(hd Y)
    fi
  od
  %
  write Z
```

```
while: if X goTo then
else: goTo fi
then:
if Y goTo then2
else2:
```

```
goTo fi2
then2:
<hd,%v1,Y,_>
<:=,Y,%v1,_>
fi2:
goTo while
fi:
```

## Exemples 3@->CPP

---

### Exemple 1

```
<nop, _, _, _>
```

```
List<BinTree> foo(List<BinTree> args)
{
    BinTree X = args.at(0), Z = args.at(1);

    //NOP

    List<BinTree> retour;
    retour.add(Y);
    return retour;
}
```

### Exemple 2

```
<:=, %v1, nil, _>
```



```
<:=, %v2, nil, _>
<cons, %v3, %v1 %v2, _>
<:=, X, %v3, _>
<:=, Z, X, _>
<nop, _, _, _>
```

```
List<BinTree> foo(List<BinTree> args)
{
    BinTree X = args.at(0), Y = args.at(1);
    List<BinTree> retour;

    X=BinTree::cons(BinTree::NIL,BinTree::NIL);
    Z=X;
    //NOP

    retour.add(Z);
    return retour;
}
```

## Exemple 3

```
<:=, %v1, nil, _>
<:=, %v2, nil, _>
<cons, %v3, %v1 %v2, _>
<:=, X, %v3, _>
<tl, %v4, X, _>
<:=, Y, %v4, _>
<:=, %v5, nil, _>
<:=, %v6, nil, _>
<cons, %v7, %v5 %v6, _>
```

```

<:=, %v8, nil, _>
<cons, %v9, %v8 %v7, _>
<hd, %v10, %v9, _>
<:=, W, %v10>
<:=, Z, X, _>
<nop, _, _, _>

```

```

List<BinTree> foo(List<BinTree> args)
{
    BinTree X = args.at(0), Y = args.at(1);

    X=BinTree::cons(1,BinTree::nil);
    Y = BinTree::tl(X);
    BinTree W = BinTree::hd(BinTree::cons(BinTree::NIL, BinTree::cons(BinTree::NIL,
BinTree::NIL)));
    Z := X;
    //NOP

    List<BinTree> retour;
    retour.add(Z);
    return retour;
}

```

## Exemple 4

```

while: if X goTo then
else: goTo fi
then:
if Y goTo then2
else2:

```

```
goTo fi2
then2:
<hd,%v1,Y,>
<:=,Y,%v1,>
fi2:
goTo while
fi:
```

```
while(BinTree::isTrue(X))
{
  if(BinTree::isTrue(Y))
  {
    Y=BinTree::hd(Y);
  }
}
```