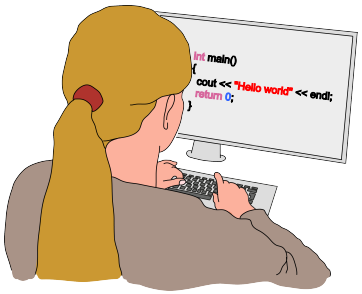
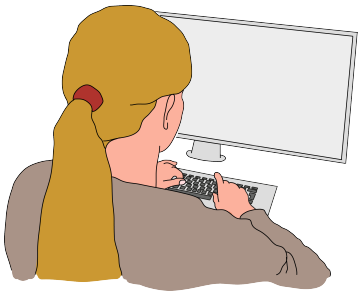
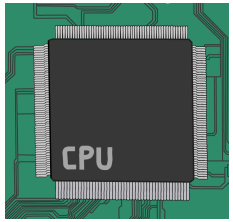
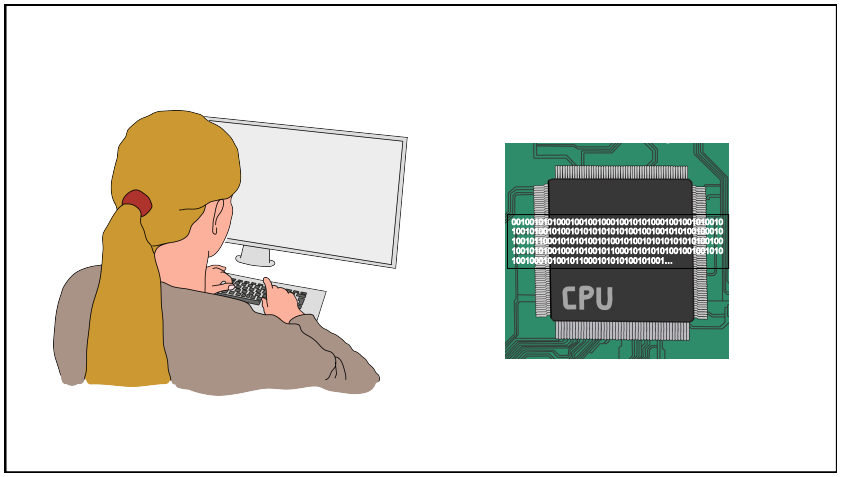
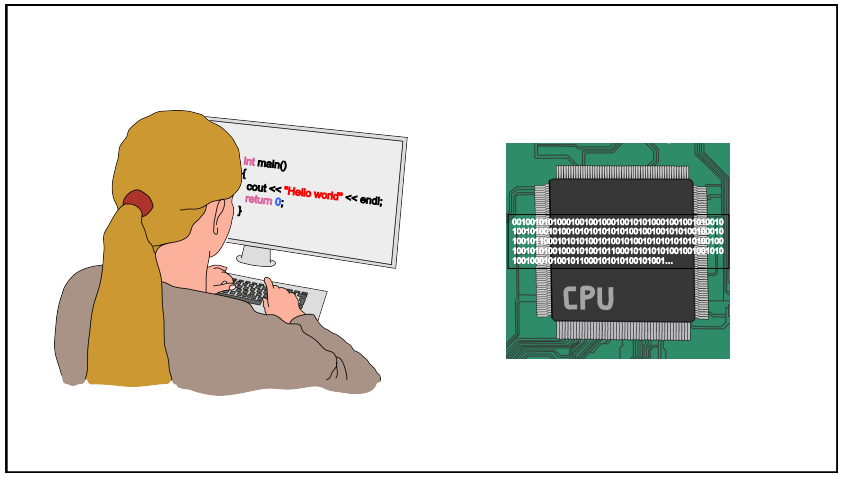
Introduction

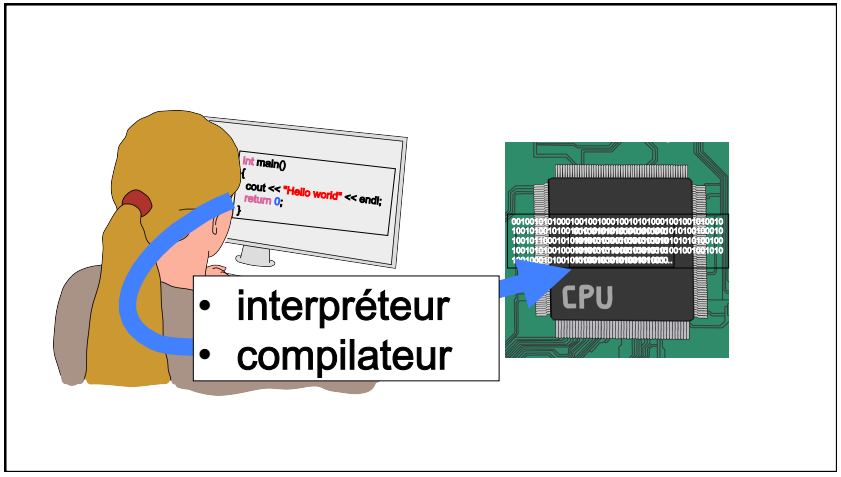


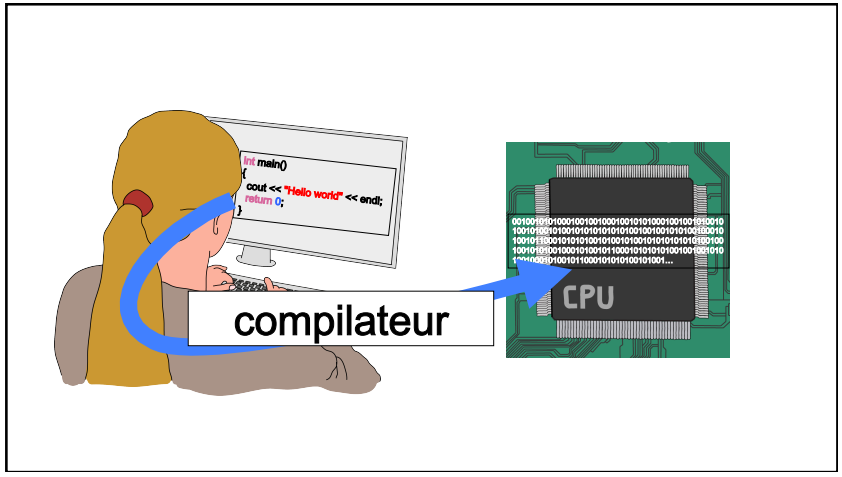


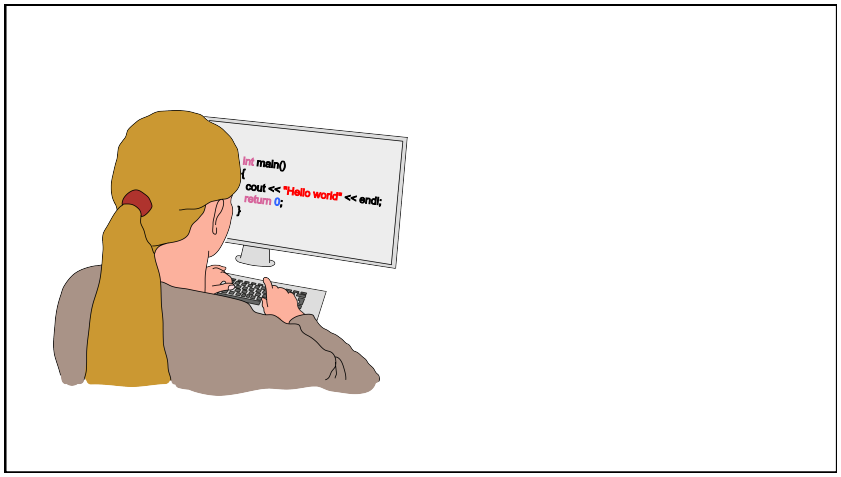


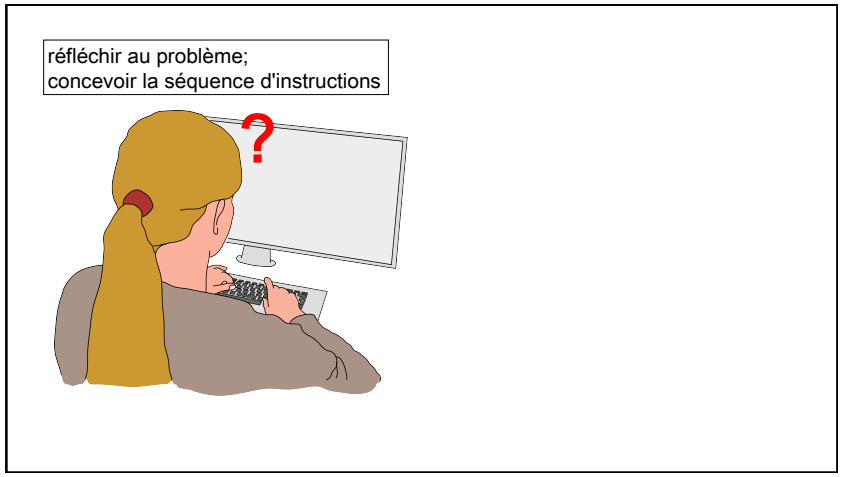

CPU

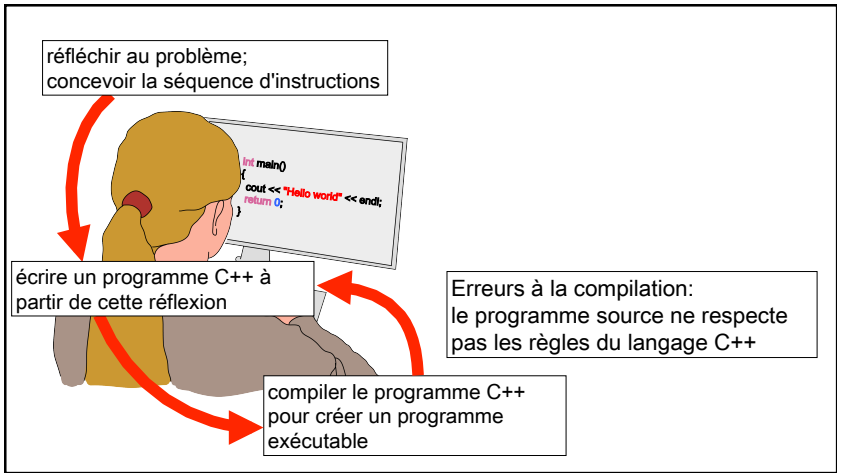

int main()
{
cout << "Hello world" << endl;
return 0;
}
CPU

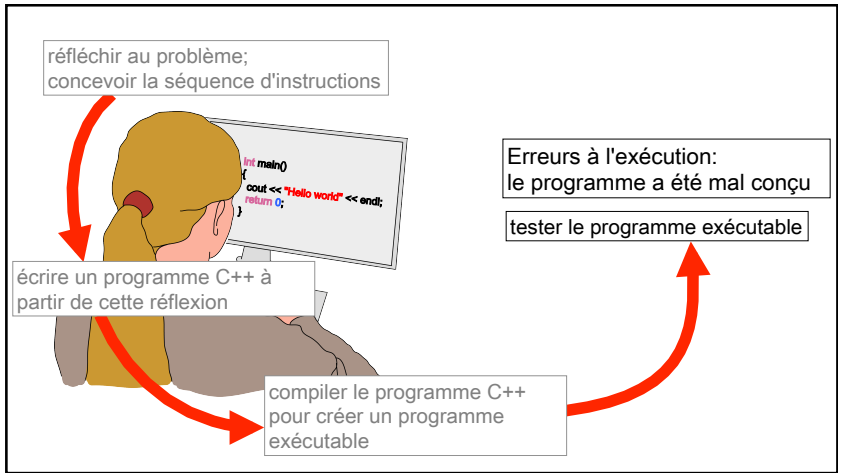

int main()
{
cout << "Hello world" << endl;
return 0;
}
• interpréteur
• compilateur
CPU

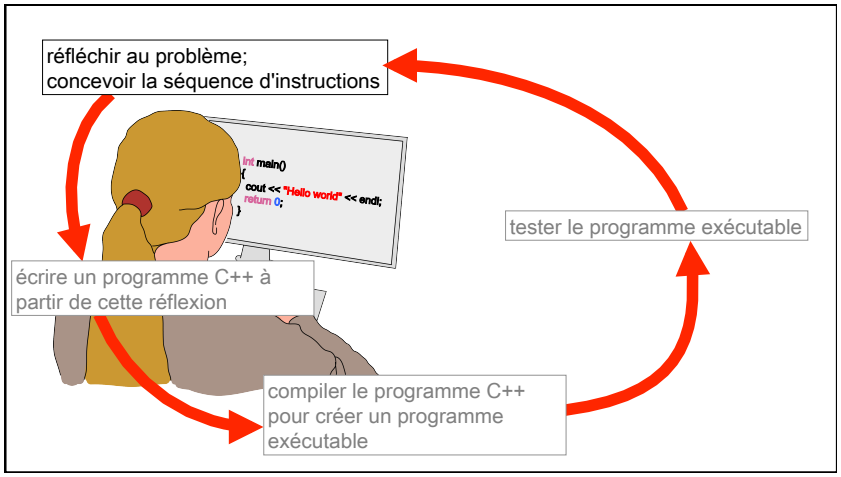

int main()
{
cout << "Hello world" << endl;
return 0;
}
compilateur
CPU


int main()
{
cout << "Hello world" << endl;
return 0;
}


réfléchir au problème;
concevoir la séquence d'instructions
?


réfléchir au problème;
concevoir la séquence d'instructions
int main()
{
cout << "Hello world" << endl;
return 0;
}
écrire un programme C++ à
partir de cette réflexion
Erreurs à la compilation:
le programme source ne respecte
pas les règles du langage C++
compiler le programme C++
pour créer un programme
exécutable


réfléchir au problème;
concevoir la séquence d'instructions
int main()
{
cout << "Hello world" << endl;
return 0;
}
écrire un programme C++ à
partir de cette réflexion
Erreurs à l'exécution:
le programme a été mal conçu
tester le programme exécutable
compiler le programme C++
pour créer un programme
exécutable



Premier programme en C++:
`Hello world`

Affichage d'un message à l'écran.

Un programme en langage C++ est un fichier texte:

```
#include <iostream>
using namespace std;

int main()
{
  cout << "Hello World!" << endl;

  return 0;
}
```

14

Ce que fait ce programme:

```
#include <iostream>
using namespace std;

int main()
{
  cout << "Hello World!" << endl;

  return 0;
}
```

```
[lepetit@cosunrays2 programmation1]$ ./hello
Hello World!
[lepetit@cosunrays2 ~]$
```

15

```
#include <iostream>
using namespace std;

int main()
{
  cout << "Hello World!" << endl;

  return 0;
}
```

Une instruction

Une instruction en C++ peut être:
- une instruction simple, terminée par un point-virgule: ';' ou
- une instruction de contrôle (condition `if`, boucle `for`…)

Les instructions sont exécutées les unes après les autres
sauf si des instructions de contrôle sont utilisées, comme nous le verrons dans la suite du cours.

16

Pour écrire à l'écran: `cout`

L'instruction:

```
cout << "Hello World!" << endl;
```

est un exemple d'affichage à l'écran à partir d'un programme C.

`endl` correspond à "retour à la ligne": ce qui sera affiché après le sera au début de la ligne suivante.

La compilation

Un programme en langage C++ est un **fichier texte**, que l'on écrit à l'aide d'un **éditeur de texte**.

Ce programme en langage C++ n'est pas exécutable directement par la machine: il doit être *compilé* pour pouvoir être exécuté par l'ordinateur.

La compilation est réalisée par un programme appelé **compilateur**. Le compilateur crée un **fichier exécutable**.

```
#include <iostream>
using namespace std;

int main()
{
  cout << "Hello World!" << endl;
  return 0;
}
```

Compilateur → Programme exécutable par l'ordinateur

Programme en langage C++:
Fichier texte compréhensible par un programmeur

Fichier compréhensible par l'ordinateur

Erreurs de programmation

Deux types d'erreurs peuvent survenir quand on écrit un programme:

1. **Les *erreurs de syntaxe*, qui surviennent à la compilation du programme:**

Un programme doit respecter précisément la syntaxe du langage C pour être accepté par le compilateur.

En cas d'erreur de syntaxe, le compilateur signale l'erreur (ou plusieurs erreurs). Par exemple, si on oublie le point-virgule à la fin du `cout`:

```
    ...
6   cout << "Hello World!" << endl
7   return 0;
}
```

le compilateur affichera le message d'erreur:

```
helloworld.cc: In function `main':
helloworld.cc:7: error: parse error before 'return' token
```

Dans ce cas, le compilateur s'arrête *sans créer d'exécutable*

2. **Les erreurs qui surviennent lors de l'exécution du programme (*bugs*):**

Le programme ne fait pas ce qui est attendu, le programme "plante",…

Savoir trouver ses erreurs

Savoir résoudre les erreurs fait partie de l'apprentissage de la programmation !

Dans le cas des erreurs de syntaxe:

- **Toujours commencer par corriger la première erreur**: les erreurs suivantes en découlent peut-être.

- Il faut savoir exploiter le message donné par le compilateur pour trouver l'erreur:
  - le compilateur indique le numéro de ligne où il *estime* que l'erreur s'est produite.
    Attention au piège: le numéro de ligne n'est qu'indicatif !
  - le compilateur décrit l'erreur qu'il a trouvé.

Le message d'erreur du compilateur

1. *Le compilateur indique le numéro de la ligne de code où il a trouvé une erreur*:

```
helloworld.cc: In function `main':
helloworld.cc:7: error: parse error before 'return' token
```

Attention, la "vraie" erreur peut être située à la ligne *précédant* celle donnée par le compilateur:

```
6   cout << "Hello World!" << endl
7   return 0;
```

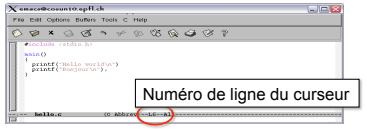
2. Il faut savoir exploiter la description de l'erreur qu'en fait le compilateur:

`helloworld.cc:7: error:` **`parse error before "return"`**

Le programme comporte une erreur avant le `return` de la ligne 7 (*parse error* = erreur de syntaxe). Ici, le compilateur trouve l'instruction `return` alors qu'il pensait trouver un point-virgule.

Exemples de messages d'erreur

Exemples de messages d'erreur:



-
```
int main()
{
  x = 10;
}
```
provoque l'erreur:
```
helloworld.cc:5: error: `x' undeclared (first use in this function)
helloworld.cc:5: error: (Each undeclared identifier is reported only once
helloworld.cc:5: error: for each function it appears in.)
```
→ la variable `x` est utilisée sans avoir été déclarée.

-
```
#include <iostrime>
```
provoque l'erreur:
```
helloworld.cc:1:21: iostrime : No such file or directory
```

-
```
#includ <iostream>
```
provoque l'erreur:
```
helloworld.cc:1:2: invalid preprocessing directive #includ
```


Exemples de messages d'erreur (2)

La ligne

```
cout << "Hello world" << end;
```


provoque, sur certains compilateurs, BEAUCOUP d'erreurs:

```
helloworld.cc(10) : error C2065: 'end' : undeclared identifier
helloworld.cc(10) : error C2593: 'operator <<' is ambiguous
        /usr/include/ostream(434): could be 'std::basic_ostream<_Elem,_Traits>::_Myt
&std::basic_ostream<_Elem,_Traits>::operator <<(std::basic_ostream<_Elem,_Traits>::_Mysb *)'
          with
          [
              _Elem=char,
              _Traits=std::char_traits<char>
          ]
    /usr/include/ostream(434): could be 'std::basic_ostream<_Elem,_Traits>::_Myt &std::basic_ostream<_Elem,_Traits>::operator
<<(std::basic_ostream<_Elem,_Traits>::_Mysb *)'
          with
          [
              _Elem=char,
              _Traits=std::char_traits<char>
          ]
```

etc...

Pas de panique ! Il faut juste changer le `end` en `endl` et toutes les erreurs disparaissent...

warning (Avertissement)

Le compilateur peut également afficher des messages d'avertissement (*warning*) quand il pense que le programme fait quelque chose de bizarre.

Ces messages ne sont pas provoqués par des erreurs de syntaxe, et le compilateur *crée* l'exécutable.

Par exemple:

```
x = x / 0;
```

est syntaxiquement valide mais provoque le *warning*:

```
helloworld.cc:7: warning: division by zero
```

En général, quand le compilateur affiche un *warning*, le programmeur a effectivement commis une erreur.

Veillez à ce que la compilation s'effectue sans l'affichage de *warning*.

Erreurs de syntaxe

Trouvez les erreurs de syntaxe de ce programme:

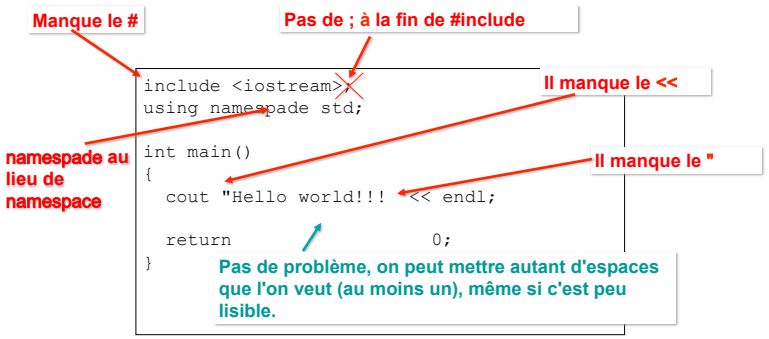
```
include <iostream>;
using namespade std;

int main()
{
  cout "Hello world!!! << endl;

  return                  0;
}
```

25

Erreurs de syntaxe




```
include <iostream>;
using namespade std;

int main()
{
  cout "Hello world!!! << endl;

  return                  0;
}
```

26

Programme corrigé

```
#include <iostream>
using namespace std;

int main()
{
  cout << "Hello world!!!" << endl;

  return 0;
}
```

27