

IFT159

Analyse et programmation

Chapitre 3 — Fonctions en C++

Gabriel Girard

Département d'informatique



15 septembre 2015

Chapitre 3 — — Fonctions en C++

- 1 Introduction
- 2 Fonctions sans paramètre
 - Syntaxe
 - Conventions
 - Séquence d'exécution
- 3 Fonctions avec paramètres
 - Syntaxe
 - Séquence d'exécution
- 4 Portée des variables
- 5 Typage
- 6 Exemple
- 7 Phases de développement avec les fonctions

Conception descendante

- 1 Introduction
- 2 Fonctions sans paramètre
 - Syntaxe
 - Conventions
 - Séquence d'exécution
- 3 Fonctions avec paramètres
 - Syntaxe
 - Séquence d'exécution
- 4 Portée des variables
- 5 Typage
- 6 Exemple
- 7 Phases de développement avec les fonctions

Abstraction procédurale

- Les fonctions permettent de faire de l'**abstraction procédurale**



Abstraction procédurale

- Les fonctions permettent de faire de
l'**abstraction procédurale**
- L'abstraction procédurale consiste à associer un nom de fonction à un algorithme complexe et à l'utiliser sans se soucier des détails d'implémentation.



Abstraction procédurale

- Les fonctions permettent de faire de
l'**abstraction procédurale**
- L'abstraction procédurale consiste à associer un nom de fonction à un algorithme complexe et à l'utiliser sans se soucier des détails d'implémentation.
- Avec l'abstraction procédurale, un programme est écrit comme une suite d'appels à des fonctions indépendantes.

Abstraction procédurale

- Les fonctions permettent de faire de
l'abstraction procédurale
- L'abstraction procédurale consiste à associer un nom de fonction à un algorithme complexe et à l'utiliser sans se soucier des détails d'implémentation.
- Avec l'abstraction procédurale, un programme est écrit comme une suite d'appels à des fonctions indépendantes.
- Avantages de l'abstraction procédurale

Abstraction procédurale

- Les fonctions permettent de faire de
l'**abstraction procédurale**
- L'abstraction procédurale consiste à associer un nom de fonction à un algorithme complexe et à l'utiliser sans se soucier des détails d'implémentation.
- Avec l'abstraction procédurale, un programme est écrit comme une suite d'appels à des fonctions indépendantes.
- Avantages de l'abstraction procédurale
 - Réutilisation

Abstraction procédurale

- Les fonctions permettent de faire de
l'abstraction procédurale
- L'abstraction procédurale consiste à associer un nom de fonction à un algorithme complexe et à l'utiliser sans se soucier des détails d'implémentation.
- Avec l'abstraction procédurale, un programme est écrit comme une suite d'appels à des fonctions indépendantes.
- Avantages de l'abstraction procédurale
 - Réutilisation
 - Masquage d'information (lisibilité)

Exemples d'analyse/conception modulaire

Écrivez un programme qui dessine une maison puis la silhouette d'une fillette sur une console.

Utilité des fonctions

- Faire abstraction de certaines étapes moins importantes

Utilité des fonctions

- Faire abstraction de certaines étapes moins importantes
- Décomposer le système pour réduire la complexité

Utilité des fonctions

- Faire abstraction de certaines étapes moins importantes
- Décomposer le système pour réduire la complexité
- Augmenter la clarté et la lisibilité

Utilité des fonctions

- Faire abstraction de certaines étapes moins importantes
- Décomposer le système pour réduire la complexité
- Augmenter la clarté et la lisibilité
- Généraliser les opérations

Utilité des fonctions

- Faire abstraction de certaines étapes moins importantes
- Décomposer le système pour réduire la complexité
- Augmenter la clarté et la lisibilité
- Généraliser les opérations
- Centraliser la définition des opérations

Utilité des fonctions

- Faire abstraction de certaines étapes moins importantes
- Décomposer le système pour réduire la complexité
- Augmenter la clarté et la lisibilité
- Généraliser les opérations
- Centraliser la définition des opérations
- Les principes des modules (en conception) s'appliquent aux fonctions

Utilité des fonctions

- Faire abstraction de certaines étapes moins importantes
- Décomposer le système pour réduire la complexité
- Augmenter la clarté et la lisibilité
- Généraliser les opérations
- Centraliser la définition des opérations
- Les principes des modules (en conception) s'appliquent aux fonctions
- D'implémenter une conception modulaire et descendante



Correspondance avec les mathématiques

- Une fonction informatique est plus générale que la fonction mathématique.

Correspondance avec les mathématiques

- Une fonction informatique est plus générale que la fonction mathématique.
- Si la fonction n'a pour entrée que des paramètres et pour sortie qu'une valeur de retour, la fonction mathématique est un bon modèle.



Correspondance avec les mathématiques

- Une fonction informatique est plus générale que la fonction mathématique.
- Si la fonction n'a pour entrée que des paramètres et pour sortie qu'une valeur de retour, la fonction mathématique est un bon modèle.
- S'il existe des interactions avec l'environnement (entrée ou sortie), la fonction mathématique n'est plus un modèle correct.

Correspondance avec les mathématiques

- Une fonction informatique est plus générale que la fonction mathématique.
- Si la fonction n'a pour entrée que des paramètres et pour sortie qu'une valeur de retour, la fonction mathématique est un bon modèle.
- S'il existe des interactions avec l'environnement (entrée ou sortie), la fonction mathématique n'est plus un modèle correct.
- Autres cas absurdes en mathématiques : une fonction sans paramètre ou une fonction sans retour.



Concept de fonctions

Le concept de fonction permet :

de regrouper des énoncés C++ dans une seule unité.

Conception descendante

- 1 Introduction
- 2 Fonctions sans paramètre**
 - Syntaxe
 - Conventions
 - Séquence d'exécution
- 3 Fonctions avec paramètres
 - Syntaxe
 - Séquence d'exécution
- 4 Portée des variables
- 5 Typage
- 6 Exemple
- 7 Phases de développement avec les fonctions

Fonctions sans paramètre — Syntaxe

Déclaration (ligne prototype)

```
type nom_fonction();
```



Fonctions sans paramètre — Syntaxe

Déclaration (ligne prototype)

```
type nom_fonction();
```

Exemple



Fonctions sans paramètre — Syntaxe

Déclaration (ligne prototype)

```
type nom_fonction();
```

Exemple

```
◇ void dessine_maison();
```



Fonctions sans paramètre — Syntaxe

Déclaration (ligne prototype)

```
type nom_fonction();
```

Exemple

- ◇ **void** dessine_maison();
- ◇ **int** mois();



Fonctions sans paramètre — Syntaxe

Déclaration (ligne prototype)

```
type nom_fonction();
```

Exemple

- ◇ **void** dessine_maison();
- ◇ **int** mois();
- ◇ **float** test2();



Fonctions sans paramètre — Syntaxe

Appel de la fonction

```
nom_fonction();
```



Fonctions sans paramètre — Syntaxe

Appel de la fonction

```
nom_fonction();
```

Exemple

Fonctions sans paramètre — Syntaxe

Appel de la fonction

```
nom_fonction();
```

Exemple

```
◇ dessine_maison();
```



Fonctions sans paramètre — Syntaxe

Appel de la fonction

```
nom_fonction();
```

Exemple

- ◇ `dessine_maison();`
- ◇ `date = mois() * nb_jour;`



Fonctions sans paramètre — Syntaxe

Appel de la fonction

```
nom_fonction();
```

Exemple

```
◇ dessine_maison();  
◇ date = mois() * nb_jour;  
◇ resultat = test2() * 3.3 / test2();
```

Fonctions sans paramètre — Syntaxe

Définition de fonction

```
/** commentaires doxygen */  
type nom_fonction()           // entête  
{  
    déclarations de fonctions // corps  
    déclarations de constantes // corps  
    déclarations de variables // corps  
    énoncés exécutables      // corps  
    return valeur;           // corps  
}
```



Fonctions sans paramètre — Conventions

Normes à respecter pour le cours :

- Noms significatifs ;



Fonctions sans paramètre — Conventions

Normes à respecter pour le cours :

- Noms significatifs ;
- Dans la définition, on place des commentaires :

Fonctions sans paramètre — Conventions

Normes à respecter pour le cours :

- Noms significatifs ;
- Dans la définition, on place des commentaires :
 - Au début (ce qu'elle fait, E/S, auteur) ;



Fonctions sans paramètre — Conventions

Normes à respecter pour le cours :

- Noms significatifs ;
- Dans la définition, on place des commentaires :
 - Au début (ce qu'elle fait, E/S, auteur) ;
 - À la fin (optionnellement) (ex. : // fin nom_fonction) ;

Fonctions sans paramètre — Conventions

Normes à respecter pour le cours :

- Noms significatifs ;
- Dans la définition, on place des commentaires :
 - Au début (ce qu'elle fait, E/S, auteur) ;
 - À la fin (optionnellement) (ex. : // fin nom_fonction) ;
- Déclaration dans la fonction qui l'utilise ;

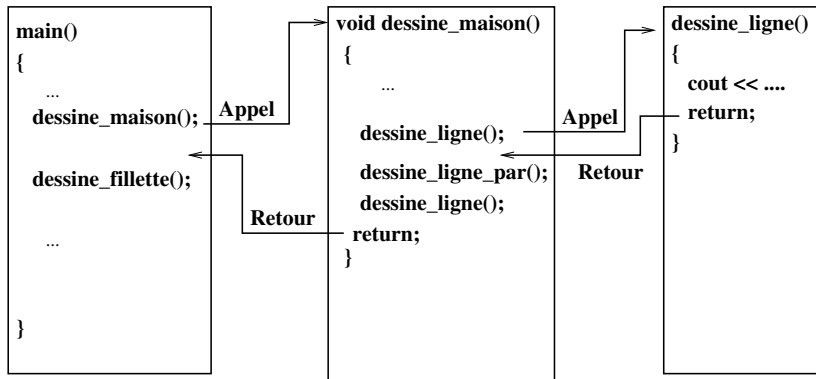
Fonctions sans paramètre — Conventions

Normes à respecter pour le cours :

- Noms significatifs ;
- Dans la définition, on place des commentaires :
 - Au début (ce qu'elle fait, E/S, auteur) ;
 - À la fin (optionnellement) (ex. : // fin nom_fonction) ;
- Déclaration dans la fonction qui l'utilise ;
- Un commentaire accompagne la déclaration.



Séquence d'exécution



Conception descendante

- 1 Introduction
- 2 Fonctions sans paramètre
 - Syntaxe
 - Conventions
 - Séquence d'exécution
- 3 Fonctions avec paramètres**
 - Syntaxe
 - Séquence d'exécution
- 4 Portée des variables
- 5 Typage
- 6 Exemple
- 7 Phases de développement avec les fonctions

Fonctions avec paramètres

- Le concept de paramètres permet de passer de l'information à une fonction.

Fonctions avec paramètres

- Le concept de paramètres permet de passer de l'information à une fonction.
- Paramètres : proches des variables d'une fonction mathématique.



Fonctions avec paramètres

Syntaxe

Déclaration (ligne prototype)

```
type nom_fonction(types_des_paramètres);
```



Fonctions avec paramètres

Syntaxe

Déclaration (ligne prototype)

```
type nom_fonction(types_des_paramètres);
```

Exemple



Fonctions avec paramètres

Syntaxe

Déclaration (ligne prototype)

```
type nom_fonction(types_des_paramètres);
```

Exemple

```
◇ void uneFonction(int, float, char, string);
```



Fonctions avec paramètres

Syntaxe

Déclaration (ligne prototype)

```
type nom_fonction(types_des_paramètres);
```

Exemple

```
◇ void uneFonction(int, float, char, string);  
◇ int mois(int);
```



Fonctions avec paramètres

Syntaxe

Déclaration (ligne prototype)

```
type nom_fonction(types_des_paramètres);
```

Exemple

```
◇ void uneFonction(int, float, char, string);  
◇ int mois(int);  
◇ float test2(float, float, float);
```



Fonctions avec paramètres

Syntaxe

Appel de fonction

```
nom_fonction(paramètres_effectifs);
```

Fonctions avec paramètres

Syntaxe

Appel de fonction

```
nom_fonction(paramètres_effectifs);
```

Exemple

Fonctions avec paramètres

Syntaxe

Appel de fonction

```
nom_fonction(paramètres_effectifs);
```

Exemple

```
◇ uneFonction(p1, p2, p3, p4);
```



Fonctions avec paramètres

Syntaxe

Appel de fonction

```
nom_fonction(paramètres_effectifs);
```

Exemple

- ◇ `uneFonction(p1, p2, p3, p4);`
- ◇ `date = mois(244) * nb_jour;`



Fonctions avec paramètres

Syntaxe

Appel de fonction

```
nom_fonction(paramètres_effectifs);
```

Exemple

```
◇ uneFonction(p1, p2, p3, p4);  
◇ date = mois(244) * nb_jour;  
◇ resultat = test2(2.2, p2, 10.0) * 3.3 /  
               test2(1.2, 3, 4.4);
```



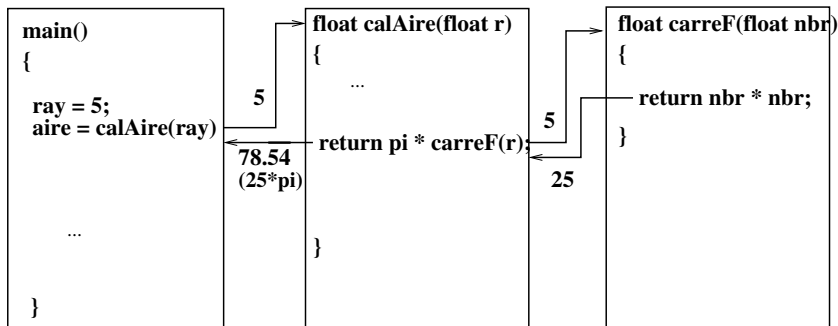
Fonctions avec paramètres

Syntaxe

Définition de fonction

```
/** commentaires doxygen */  
type nom_fct(paramètres formels) // entête  
{  
    déclarations de fonctions // corps  
    déclarations de constantes // corps  
    déclarations de variables // corps  
    énoncé exécutables // corps  
    return valeur; // corps  
}
```

Séquence d'exécution



Conception descendante

- 1 Introduction
- 2 Fonctions sans paramètre
 - Syntaxe
 - Conventions
 - Séquence d'exécution
- 3 Fonctions avec paramètres
 - Syntaxe
 - Séquence d'exécution
- 4 Portée des variables**
- 5 Typage
- 6 Exemple
- 7 Phases de développement avec les fonctions



Portée des variables et constantes

- Les identificateurs déclarés à l'intérieur d'une fonction sont appelés des identificateurs locaux et ils n'existent que pour cette fonction.

Portée des variables et constantes

- Les identificateurs déclarés à l'intérieur d'une fonction sont appelés des identificateurs locaux et ils n'existent que pour cette fonction.
- Les paramètres formels n'existent qu'à l'intérieur de la fonction.

Portée des variables et constantes

- Les identificateurs déclarés à l'intérieur d'une fonction sont appelés des identificateurs locaux et ils n'existent que pour cette fonction.
- Les paramètres formels n'existent qu'à l'intérieur de la fonction.
- Les identificateurs déclarés à l'extérieur des fonctions sont appelés des identificateurs globaux et ils existent pour toutes les fonctions.

Portée des variables et constantes

- Les identificateurs déclarés à l'intérieur d'une fonction sont appelés des identificateurs locaux et ils n'existent que pour cette fonction.
- Les paramètres formels n'existent qu'à l'intérieur de la fonction.
- Les identificateurs déclarés à l'extérieur des fonctions sont appelés des identificateurs globaux et ils existent pour toutes les fonctions.
 - On peut utiliser des constantes globales.

Portée des variables et constantes

- Les identificateurs déclarés à l'intérieur d'une fonction sont appelés des identificateurs locaux et ils n'existent que pour cette fonction.
- Les paramètres formels n'existent qu'à l'intérieur de la fonction.
- Les identificateurs déclarés à l'extérieur des fonctions sont appelés des identificateurs globaux et ils existent pour toutes les fonctions.
 - On peut utiliser des constantes globales.
 - Cela évite des répétitions inutiles.

Conception descendante

- 1 Introduction
- 2 Fonctions sans paramètre
 - Syntaxe
 - Conventions
 - Séquence d'exécution
- 3 Fonctions avec paramètres
 - Syntaxe
 - Séquence d'exécution
- 4 Portée des variables
- 5 Typage**
- 6 Exemple
- 7 Phases de développement avec les fonctions

Typage

- C++ est un langage fortement typé.

Typage

- C++ est un langage fortement typé.
 - ⇒ Il associe un type à chaque variable ou constante.

Typage

- C++ est un langage fortement typé.
 - ⇒ Il associe un type à chaque variable ou constante.
 - ⇒ La variable doit être utilisée d'une façon consistante à son type.

Typage

- C++ est un langage fortement typé.
 - ⇒ Il associe un type à chaque variable ou constante.
 - ⇒ La variable doit être utilisée d'une façon consistante à son type.
 - ⇒ Il se produira des erreurs s'il y a des incohérences.

Typage

- C++ est un langage fortement typé.
 - ⇒ Il associe un type à chaque variable ou constante.
 - ⇒ La variable doit être utilisée d'une façon consistante à son type.
 - ⇒ Il se produira des erreurs s'il y a des incohérences.
- Changement explicite de type → *type (nomVariable)*.

Typage

- C++ est un langage fortement typé.
 - ⇒ Il associe un type à chaque variable ou constante.
 - ⇒ La variable doit être utilisée d'une façon consistante à son type.
 - ⇒ Il se produira des erreurs s'il y a des incohérences.
- Changement explicite de type → *type (nomVariable)*.
- On peut aussi retrouver → *(type) nomVariable*.

Typage

- C++ est un langage fortement typé.
 - ⇒ Il associe un type à chaque variable ou constante.
 - ⇒ La variable doit être utilisée d'une façon consistante à son type.
 - ⇒ Il se produira des erreurs s'il y a des incohérences.
- Changement explicite de type → *type (nomVariable)*.
- On peut aussi retrouver → *(type) nomVariable*.
- On appelle ceci une conversion explicite ou un *casting*.

Conception descendante

- 1 Introduction
- 2 Fonctions sans paramètre
 - Syntaxe
 - Conventions
 - Séquence d'exécution
- 3 Fonctions avec paramètres
 - Syntaxe
 - Séquence d'exécution
- 4 Portée des variables
- 5 Typage
- 6 Exemple
- 7 Phases de développement avec les fonctions

Exemple de correspondance math-info

La fonction mathématique

$$\begin{aligned} f : \mathbb{N} &\rightarrow \mathbb{R} \\ x &\mapsto x \times 2.33 \end{aligned}$$

se traduit par la fonction informatique

```
float f ( int x )  
{  
    return x * 2.33 ;  
}
```



Exemple de correspondance math-info

La fonction mathématique

$$\begin{aligned} f : \mathbb{N} \times \mathbb{R} &\rightarrow \mathbb{R} \\ (x, y) &\mapsto xy + 1 \end{aligned}$$

se traduit par la fonction informatique

```
float f ( int x, float y )  
{  
    return x*y+1 ;  
}
```

Exemple de correspondance math-info

La fonction mathématique

$$\lfloor \cdot \rfloor : \mathbb{R} \rightarrow \mathbb{N}$$
$$x \mapsto n \in \mathbb{N} \text{ tel que } n \leq x < n + 1$$

se traduit par la fonction informatique

```
int partie_entiere ( float y )
{
    return (int) floor(y);
}
```

Module « ratio par tête »

Entrée :

Module « ratio par tête »

Entrée :

- (*paramètre*) N_p , Nbre de personnes (*entier*)

Module « ratio par tête »

Entrée :

- (*paramètre*) N_p , Nbre de personnes (*entier*)
- (*paramètre*) C , Charge du serveur (*réel*)

Module « ratio par tête »

Entrée :

- (*paramètre*) N_p , Nbre de personnes (*entier*)
- (*paramètre*) C , Charge du serveur (*réel*)

Sortie : (*valeur de retour*) R , le ratio par tête (*réel*)

Module « ratio par tête »

Entrée :

- (*paramètre*) N_p , Nbre de personnes (*entier*)
- (*paramètre*) C , Charge du serveur (*réel*)

Sortie : (*valeur de retour*) R , le ratio par tête (*réel*)

Formules :



Module « ratio par tête »

Entrée :

- (*paramètre*) N_p , Nbre de personnes (*entier*)
- (*paramètre*) C , Charge du serveur (*réel*)

Sortie : (*valeur de retour*) R , le ratio par tête (*réel*)

Formules :

$$1 \quad R = \frac{T}{N_p}$$

Module « ratio par tête »

Entrée :

- (*paramètre*) N_p , Nbre de personnes (*entier*)
- (*paramètre*) C , Charge du serveur (*réel*)

Sortie : (*valeur de retour*) R , le ratio par tête (*réel*)

Formules :

1 $R = \frac{T}{N_p}$

2 la tension $T = C^2 \sinus(C)$

Module « ratio par tête »

- Algorithme :

Module « ratio par tête »

■ Algorithme :

- 1 Calculer le carré de la charge du serveur.

Module « ratio par tête »

■ Algorithme :

- 1 Calculer le carré de la charge du serveur.
- 2 Calculer le sinus de la charge (Module « Calculer sinus »).

Module « ratio par tête »

■ Algorithme :

- 1 Calculer le carré de la charge du serveur.
- 2 Calculer le sinus de la charge (Module « Calculer sinus »).
- 3 Multiplier les deux précédents résultats (formule 2).

Module « ratio par tête »

■ Algorithme :

- 1 Calculer le carré de la charge du serveur.
- 2 Calculer le sinus de la charge (Module « Calculer sinus »).
- 3 Multiplier les deux précédents résultats (formule 2).
- 4 Le ratio est le résultat précédent divisé par le nombre de personnes (formule 1).

Fonction implémentant « ratio par tête »

Voici la fonction en C++

```
float ratio_par_tete (int nbre_personnes, float charge)
{
    // declaration de la fonction appelee
    float calculer_sinus (float);

    // declaration des variables
    float ratio; // sortie

    float tension;
    float carre_charge;
    float sinus_charge;
```

Fonction implémentant « ratio par tête »

```
// Debut du cours de la fonction  
// 1. Calculer le carré de la charge du serveur.  
carre_charge = charge * charge ;  
  
// 2. Calculer le sinus de la charge.  
sinus_charge = calculer_sinus(charge);  
  
// 3. Multiplier les deux précédents résultats.  
tension = carre_charge * sinus_charge;  
  
// 4. Le ratio est le résultat précédent divisé  
//    par le nombre de personnes.  
ratio = tension / nbre_personnes;  
  
return ratio;
```

}

Conception descendante

- 1 Introduction
- 2 Fonctions sans paramètre
 - Syntaxe
 - Conventions
 - Séquence d'exécution
- 3 Fonctions avec paramètres
 - Syntaxe
 - Séquence d'exécution
- 4 Portée des variables
- 5 Typage
- 6 Exemple
- 7 Phases de développement avec les fonctions

Phases de développement

1 Comprendre le problème

Phases de développement

- 1 Comprendre le problème
- 2 Faire analyse du problème (entrées, sorties, formules, ...)

Phases de développement

- 1 Comprendre le problème
- 2 Faire analyse du problème (entrées, sorties, formules, ...)
- 3 Décomposition du problème en sous-problèmes. On identifie les modules.



Phases de développement

- 1 Comprendre le problème
- 2 Faire analyse du problème (entrées, sorties, formules, ...)
- 3 Décomposition du problème en sous-problèmes. On identifie les modules.
 - 1 Conception du problème principal (algorithme exprimé en termes d'appels à des modules).



Phases de développement

- 1 Comprendre le problème
- 2 Faire analyse du problème (entrées, sorties, formules, ...)
- 3 Décomposition du problème en sous-problèmes. On identifie les modules.
 - 1 Conception du problème principal (algorithme exprimé en termes d'appels à des modules).
 - 2 Pour chaque sous-problème (module) on refait la conception.



Phases de développement

- 1 Comprendre le problème
- 2 Faire analyse du problème (entrées, sorties, formules, ...)
- 3 Décomposition du problème en sous-problèmes. On identifie les modules.
 - 1 Conception du problème principal (algorithme exprimé en termes d'appels à des modules).
 - 2 Pour chaque sous-problème (module) on refait la conception.
- 4 On code séparément chacun des modules grâce aux fonctions.

Phases de développement

- 1 Comprendre le problème
- 2 Faire analyse du problème (entrées, sorties, formules, ...)
- 3 Décomposition du problème en sous-problèmes. On identifie les modules.
 - 1 Conception du problème principal (algorithme exprimé en termes d'appels à des modules).
 - 2 Pour chaque sous-problème (module) on refait la conception.
- 4 On code séparément chacun des modules grâce aux fonctions.
- 5 On teste séparément chaque fonction.

Phases de développement

- 1 Comprendre le problème
- 2 Faire analyse du problème (entrées, sorties, formules, ...)
- 3 Décomposition du problème en sous-problèmes. On identifie les modules.
 - 1 Conception du problème principal (algorithme exprimé en termes d'appels à des modules).
 - 2 Pour chaque sous-problème (module) on refait la conception.
- 4 On code séparément chacun des modules grâce aux fonctions.
- 5 On teste séparément chaque fonction.
- 6 On intègre et teste les fonctions.

Erreurs célèbres

■ Jupiter ou Viking - mission vers mars

On pouvait télécharger de nouveaux programmes en mémoire pour y implanter de nouvelles fonctionnalités. Après plusieurs jours d'exploration, la mémoire faisait défaut. On décida de détruire le module de contrôle de l'atterrissage. Aussitôt ce module effacé, on perdit tout contact avec la sonde. Le module de contrôle d'atterrissage contenait une partie ou toutes les fonctions de navigation stellaire qu'utilisait aussi le module d'alignement de l'antenne.

Erreurs célèbres

- Jupiter ou Viking - mission vers mars
On pouvait télécharger de nouveaux programmes en mémoire pour y implanter de nouvelles fonctionnalités. Après plusieurs jours d'exploration, la mémoire faisait défaut. On décida de détruire le module de contrôle de l'atterrissage. Aussitôt ce module effacé, on perdit tout contact avec la sonde. Le module de contrôle d'atterrissage contenait une partie ou toutes les fonctions de navigation stellaire qu'utilisait aussi le module d'alignement de l'antenne.
- F18 : ordinateur ouvrait et refermait trop rapidement la pince de retenue des missiles.

Conclusion

- Anonymous
It's not a bug – it's an undocumented feature.



Conclusion

- Anonymous

It's not a bug – it's an undocumented feature.

- Jeff Atwood

“In software, we rarely have meaningful requirements. Even if we do, the only measure of success that matters is whether our solution solves the customer's shifting idea of what their problem is.”

Exercice

- 1 Écrire un programme qui calcule la durée du trajet d'un projectile et son altitude au dessus du sol lorsqu'il atteint la cible. Le programme prend donc en entrée l'angle du tir (en degrés), la distance de la cible et la vitesse du projectile au départ.

$$\text{Formule pour la durée du trajet : } \mathit{duree} = \frac{\mathit{distance}}{\mathit{vitesse} \times \cos(\mathit{radian})}$$

Formule pour l'altitude :

$$\mathit{altitude} = \mathit{vitesse} \times \sin(\mathit{radian}) \times \mathit{duree} - \frac{g \times \mathit{duree}^2}{2}$$

$$\text{Conversion de degré à radian : } \mathit{radian} = \frac{\mathit{angle} \times \mathit{PI}}{180}$$

Exercice

- 1 Écrire un programme qui calcule la durée du trajet d'un projectile et son altitude au dessus du sol lorsqu'il atteint la cible. Le programme prend donc en entrée l'angle du tir (en degrés), la distance de la cible et la vitesse du projectile au départ.

$$\text{Formule pour la durée du trajet : } \mathit{duree} = \frac{\mathit{distance}}{\mathit{vitesse} \times \cos(\mathit{radian})}$$

Formule pour l'altitude :

$$\mathit{altitude} = \mathit{vitesse} \times \sin(\mathit{radian}) \times \mathit{duree} - \frac{g \times \mathit{duree}^2}{2}$$

$$\text{Conversion de degré à radian : } \mathit{radian} = \frac{\mathit{angle} \times \mathit{PI}}{180}$$

- 2 Écrire une fonction qui calcule le volume d'une sphère de rayon réel donné. Écrire aussi le programme principal qui utilise cette fonction.