Initiation à l'algorithmique et à la programmation en C

Tout le catalogue sur
www.dunod.com

ÉDITEUR DE SAVOIRS

Rémy Malgouyres
Rita Zrour
Fabien Feschet

Initiation à l'algorithmique et à la programmation en C

Cours avec 129 exercices corrigés

3e édition

DUNOD

Illustration de couverture : © alwyncooper - istock.com

Le pictogramme qui figure ci-contre mérite une explication. Son objet est d'alerter le lecteur sur la menace que représente pour l'avenir de l'écrit, particulièrement dans le domaine de l'édition technique et universitaire, le développement massif du photocopillage.
Le Code de la propriété intellectuelle du 1er juillet 1992 interdit en effet expressément la photocopie à usage collectif sans autorisation des ayants droit. Or, cette pratique s'est généralisée dans les établissements d'enseignement supérieur, provoquant une baisse brutale des achats de livres et de revues, au point que la possibilité même pour les auteurs de créer des œuvres nouvelles et de les faire éditer correctement est aujourd'hui menacée.
Nous rappelons donc que toute reproduction, partielle ou totale, de la présente publication est interdite sans autorisation de l'auteur, de son éditeur ou du Centre français d'exploitation du droit de copie (CFC, 20, rue des Grands-Augustins, 75006 Paris).

© Dunod, 2008, 2011, 2014

5 rue Laromiguière, 75005 Paris
www.dunod.com

ISBN 978-2-10-071001-0

Le Code de la propriété intellectuelle n'autorisant, aux termes de l'article L. 122-5, 2° et 3° a), d'une part, que les « copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective » et, d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration, « toute représentation ou reproduction intégrale ou partielle faite sans le consentement de l'auteur ou de ses ayants droit ou ayants cause est illicite » (art. L. 122-4).
Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait donc une contrefaçon sanctionnée par les articles L. 335-2 et suivants du Code de la propriété intellectuelle.

TABLE DES MATIÈRES

© Dunod. Toute reproduction non autorisée est un délit.

© Dunod. Toute reproduction non autorisée est un délit.

© Dunod. Toute reproduction non autorisée est un délit.

Annexes

© Dunod. Toute reproduction non autorisée est un délit.

AVANT-PROPOS

QUE CONTIENT CE LIVRE ?

Cet ouvrage est destiné aux étudiants de première année des filières informatique (L1, DUT, certaines licences professionnelles), et à tous ceux qui veulent acquérir des bases solides en programmation, sans connaissances préalables, avec la volonté d'approfondir. Il inclut la programmation en langage *C* (syntaxe, exécution conditionnelle, boucles itératives, tableaux, fichiers, allocation dynamique de mémoire, récursivité...), les algorithmes et la complexité (langage algorithmique, complexité d'algorithmes, tris...), ainsi que les structures de données (listes chaînées, piles, files, arbres, graphes et parcours de graphes). L'un des objectifs fixés lors de la rédaction de ce livre était de produire un ouvrage digeste. Le texte ne vise pas l'exhaustivité sur le langage *C*.

À L'ÉTUDIANT

Ce livre permet d'aborder la programmation en langage *C* sans connaissance préalable de l'informatique. Il est conçu pour pouvoir être utilisé comme un outil d'apprentissage en autodidacte. L'étudiant peut utiliser ce livre, et notamment les exercices corrigés, en complément de l'enseignement qu'il reçoit par ailleurs. Il peut aussi apprendre seul, en abordant les chapitres dans l'ordre, en contrôlant ses connaissances avec les exercices corrigés et avec les travaux pratiques sur machine.

Le texte présente de manière concise les bases qui sont nécessaires aux exercices. Les exercices de chaque chapitre sont progressifs et doivent de préférence être traités dans l'ordre. Une indication de la difficulté des exercices est donnée par des étoiles lors de l'énoncé.

Les pièges et les erreurs les plus courants sont mis en évidence clairement dans le texte par des panneaux spéciaux *Attention !*

Compléments

Des compléments sont donnés au fil du texte ; ils apportent un éclairage sur certaines notions difficiles et peuvent être abordés lors d'une seconde lecture ou lors de la phase d'exercices.

© Dunod. Toute reproduction non autorisée est un délit.

Des annexes expliquent comment compiler un programme *C* sur son ordinateur personnel avec le compilateur gratuit `gcc`. Des sujets supplémentaires de travaux pratiques sur machine pour chaque chapitre sont disponibles sur le site web de l'auteur Rémy Malgouyres `http://www.malgouyres.org/`.

À L'ENSEIGNANT

Le langage *C*, dont découlent de très nombreux autres langages actuels, reste la référence pour la programmation bas niveau. En particulier, ce langage est très présent en programmation système et réseaux. L'apprentissage de l'algorithmique en *C* permet notamment de percevoir la gestion mémoire à partir d'une gestion manuelle, qui apporte une meilleure compréhension des constructeurs et destructeurs du *C++* ou des subtilités du *garbage collector* en *Java*... L'apprentissage des structures de données en *C* permet de comprendre en détail la nature des objets manipulés à travers les librairies *Java* ou la *STL* du *C++*. L'abord de la programmation par le *C* est le moyen le plus efficace de former des informaticiens complets, ayant au final une maîtrise parfaite du bas niveau comme du haut niveau et une compréhension profonde des concepts.

L'enseignant pourra s'appuyer sur la structure de l'ouvrage, qui permet de dérouler le contenu sans référence à ce qui suit. L'exposé, mais surtout les exercices contiennent de très nombreux exemples qui peuvent être réutilisés. L'ouvrage est conçu pour un apprentissage autonome. Les exercices de chaque chapitre sont progressifs. En complément des exercices propres à chaque enseignant, les exercices du livre peuvent être donnés à faire aux étudiants qui peuvent consulter la solution *a posteriori*.

Partie 1

Bases du langage C

1 Qu'est-ce qu'un ordinateur ?

1.1 Exemples d'applications de l'informatique

Voici quelques exemples d'utilisation des ordinateurs :

- **Bureautique** L'ordinateur emmagasine des données saisies par une secrétaire par exemple (textes, chiffres, fichiers clients, etc.) ou des données issues d'archives, et les met en forme pour permettre une compréhension synthétique, un affichage ou une communication de ces données.
- **Jeux vidéo** L'ordinateur combine des données entrées par le concepteur du jeu (données sur l'univers) avec les événements créés par l'utilisateur du jeu (clics de souris, etc.) pour générer des images, du son, etc.
- **Prévision météorologique** À partir de la donnée des relevés de toutes les stations météo d'une zone géographique, l'ordinateur calcule une situation future et génère des cartes de températures et de pressions atmosphériques.
- **Applications multimédia sur Internet** L'ordinateur *télécharge* des données stockées sur un *serveur* distant et affiche ces données sur l'ordinateur de l'utilisateur. Éventuellement, des actions de l'utilisateur peuvent influer sur les données affichées (on parle alors d'applications interactives).

Dans tous ces exemples, l'ordinateur *traite des données*, et produit un résultat, soit communiqué à l'utilisateur (son, images, texte), soit affiché sur un écran, ou stocké sur un disque, ou un autre support.

1.2 Codage des données

Les données informatiques sont toujours, en fin de compte, **codées en binaire**, c'est-à-dire qu'elles sont représentées par des suites de 0 et de 1. En effet, les données binaires sont plus faciles à mémoriser sur des supports physiques (bandes magnétiques, disques, etc.).

Par exemple, si l'on veut stocker un nombre entier sur le disque dur d'un ordinateur, on code généralement ce nombre en base 2 au lieu de le coder en base 10 comme nous y sommes naturellement habitués.

© Dunod. Toute reproduction non autorisée est un délit.

Ainsi le nombre 12 (en base 10) sera codé en base 2 par la suite binaire 00001100, ce qui signifie que :

$$\begin{aligned} 12 &= 0 + 0 + 0 + 0 + 8 + 4 + 0 + 0 \\ &= \mathbf{0} \times 2^7 + \mathbf{0} \times 2^6 + \mathbf{0} \times 2^5 + \mathbf{0} \times 2^4 + \mathbf{1} \times 2^3 + \mathbf{1} \times 2^2 + \mathbf{0} \times 2^1 + \mathbf{0} \times 2^0 \end{aligned}$$

Une donnée égale soit à 0 soit à 1 s'appelle un *bit*. Une séquence de 8 bits consécutifs s'appelle un *octet* (en anglais *byte*). On mesure la quantité de mémoire stockée dans les ordinateurs en :

- Octets : 1 octet = 8 bits ;
- Kilo-octets (en abrégé *Ko* ou en anglais *Kb*) : un *Ko* vaut 1024 octets.
- Méga-octets (en abrégé *Mo* ou *Mb*) : un *Mo* vaut 1 048 576 octets
- Giga-octets (en abrégé *Go* ou *Gb*) : un *Go* vaut 1 073 741 824 octets

L'apparition des nombres 1024, 1 048 576 et 1 073 741 824 peut paraître surprenante, mais ce sont des puissances de 2. On retient en général qu'un *Ko* fait environ mille octets, un *Mo* environ un million, et un *Go* environ un milliard.

1.3 Fonctionnement d'un ordinateur

1.3.1 Système d'exploitation

Un programme informatique doit recevoir des données pour les traiter et produire d'autres données. Pour que le programme puisse fonctionner, il faut du matériel (composants électroniques), et il faut une couche logicielle intermédiaire avec le matériel, appelée *système d'exploitation*. Le système assure la communication entre le programme informatique et le matériel, et permet au programme d'agir sur le matériel.

1.3.2 Processeur

Le processeur effectue des opérations (par exemple des opérations arithmétiques comme des additions ou des multiplications). Ces opérations sont *câblées* dans le processeur, c'est-à-dire qu'elles sont effectuées par des circuits électroniques pour être efficaces. Avec le temps, de plus en plus d'opérations complexes sont câblées au niveau du processeur, ce qui augmente l'efficacité. La vitesse d'un processeur, c'est-à-dire en gros le nombre d'opérations par seconde, appelée *vitesse d'horloge*, est mesurée en hertz (Hz), kilohertz ($1kHz = 1000Hz$) , megahertz ($1MHz = 10^6Hz$, et gigahertz ($1GHz = 10^9Hz$). Sur les architectures récentes, la puce contient plusieurs *cores*, chaque core étant l'équivalent d'un processeur et les cores communiquant entre eux très rapidement par des *bus* de données. Pour la personne qui programme en *C*, la configuration et la structure de la puce est *transparente*, c'est-à-dire que l'on n'a pas à s'en préoccuper (sauf pour l'optimisation en programmation très avancée).

1.3.3 Mémoire centrale

Au cours du déroulement du programme, celui-ci utilise des données, soit les données fournies en entrée, soit des données intermédiaires que le programme utilise pour fonctionner. Ces données sont stockées dans des *variables*. Physiquement, les variables sont des données binaires dans la *mémoire centrale* (appelée aussi mémoire *RAM*). La mémoire centrale communique rapidement avec le processeur. Lorsque le processeur effectue un calcul, le programmeur peut indiquer que le résultat de ce calcul doit être mémorisé dans une variable (en *RAM*). Le processeur pourra accéder plus tard au contenu de cette variable pour effectuer d'autres calculs ou produire un résultat en sortie. La quantité de mémoire *RAM* est mesurée en octets (ou en mégaoctets ou gigaoctets). Les données en mémoire centrale ne sont conservées que pendant le déroulement du programme, et disparaissent lorsque le programme se termine (notamment lorsque l'on éteint l'ordinateur).

1.3.4 Périphériques

Le programme reçoit des données des périphériques en entrée, et communique ses résultats en sortie à des périphériques. Une liste (non exhaustive) de périphériques usuels est :

- le clavier qui permet à l'utilisateur de saisir du texte ;
- la souris qui permet à l'utilisateur de sélectionner, d'activer ou de créer à la main des objets graphiques ;
- l'écran qui permet aux programmes d'afficher des données sous forme graphique ;
- l'imprimante qui permet de sortir des données sur support papier ;
- le disque dur ou la clef *USB* qui permettent de stocker des données de manière permanente. Les données sauvegardées sur un tel disque sont préservées, y compris après terminaison du programme ou lorsque l'ordinateur est éteint, contrairement aux données stockées en mémoire centrale qui disparaissent lorsque le programme se termine.

Les périphériques d'entrée (tels que le clavier et la souris) transmettent les données dans un seul sens, du périphérique vers la mémoire centrale. Les périphériques de sortie (tels que l'écran ou l'imprimante) recoivent des données dans un seul sens, de la mémoire centrale (ou de la mémoire vidéo) vers le périphérique. Les périphériques d'entrée-sortie (tels que le disque dur, le port USB, ou la carte réseau) permettent la communication dans les deux sens entre la mémoire centrale et le périphérique.

© Dunod. Toute reproduction non autorisée est un délit.

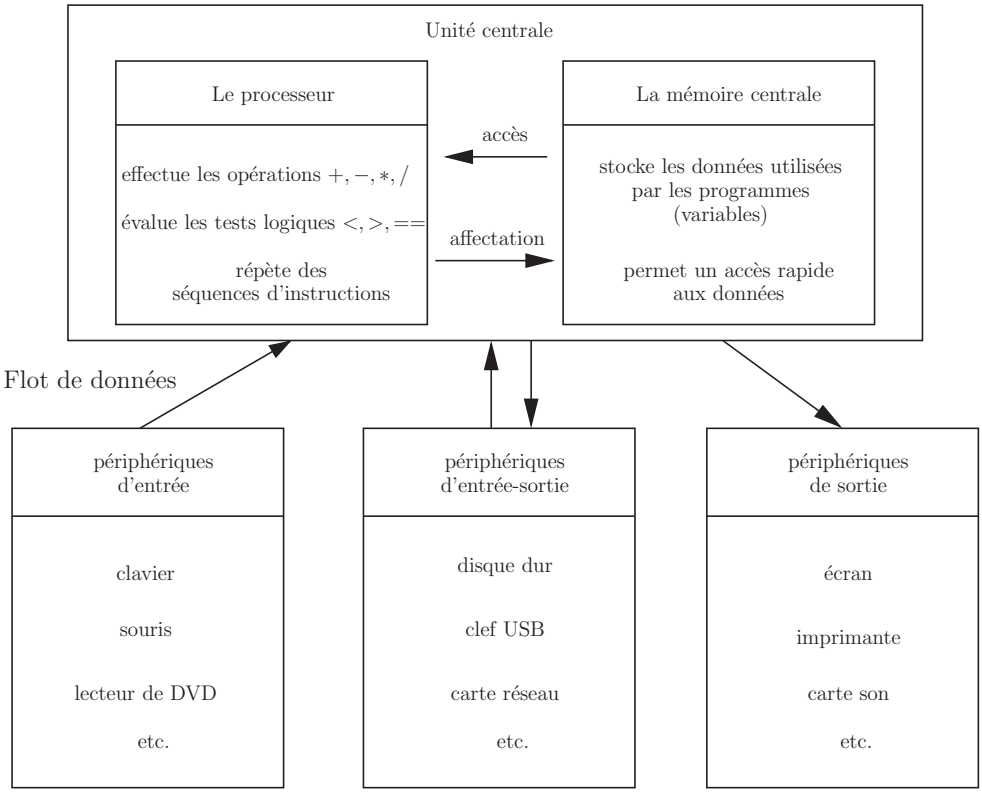

Figure 1.1 – Schéma d'architecture d'un ordinateur.

Retenir l'essentiel

Un ordinateur a pour fonction de stocker, traiter et communiquer des informations, appelées *données*. Les données sont stockées en binaire sur les disques, et transmises vers des périphériques d'entrée et/ou de sortie. Le système d'exploitation assure la communication entre le matériel ou les périphériques et les programmes, qui définissent et appliquent le traitement des données. Les programmes, écrits dans un certain langage, comme le *C*, s'appuient sur la mémoire *RAM*, qui permet de stocker temporairement des données nécessaires aux calculs, qui sont eux effectués par le processeur.

PREMIERS PROGRAMMES

2

2.1 QU'EST-CE QU'UN PROGRAMME ?

Un programme informatique réalise en général trois choses :

- **Il lit des données en entrée**. Le programme doit en effet savoir à partir de quoi travailler. Par exemple, pour utiliser une calculatrice, on doit lui donner des nombres et lui dire quelles opérations effectuer. Pour cela, on utilise souvent un *clavier*, mais le programme peut aussi tirer les données d'un *disque dur* ou encore d'un *autre ordinateur via* un réseau ou autre.
- **Il effectue des calculs**. À partir des données en entrée, le programme va appliquer automatiquement des méthodes pour traiter ces données et produire un résultat. Les méthodes que sont capables d'effectuer les ordinateurs s'appellent des *algorithmes*. Par exemple, une calculatrice va appliquer l'algorithme d'addition ou de multiplication.
- **Il écrit des données en sortie**. Lorsque le programme a obtenu un résultat, il doit écrire ce résultat quelque part pour qu'on puisse l'utiliser. Par exemple, une calculatrice va afficher un résultat à l'*écran* ou stocker le résultat en *mémoire*.

Le travail d'un programmeur consiste à créer des programmes informatiques. Le programmeur doit pour cela expliquer à l'ordinateur dans un certain langage, appelé *langage de programmation*, quelles sont les données et quelles sont les méthodes à appliquer pour traiter ces données. Dans ce chapitre, nous verrons en langage *C*, les premiers exemples permettant :

1. de lire une donnée au clavier avec la fonction `scanf` ;
2. d'effectuer les calculs les plus simples sur des nombres et de stocker le résultat dans une variable ;
3. d'afficher un texte ou un nombre à l'écran avec la fonction `printf`.

Ce faisant, nous verrons la structure d'un programme *C* très simple et quelques notions sur la *syntaxe* du langage. Les notions vues dans ces exemples seront développées dans les chapitres suivants. Une fois que le programmeur a écrit son programme, qui est du texte en langage *C*, il doit *compiler* le programme pour créer un fichier *exécutable* pour qu'un utilisateur du programme puisse utiliser ce programme. Le processus de compilation est décrit en annexe.

© Dunod. Toute reproduction non autorisée est un délit.

2.2 Afficher un mot

Voici un programme *C* qui écrit un message de bienvenue : le mot « Bonjour ».

```
#include <stdio.h>          /* pour pouvoir lire et écrire */

int main(void)              /* programme principal */
{
   printf("Bonjour !\n");      /* écriture à l'écran */
   return 0;
}
```

Les phrases comprises entre /∗ et ∗/ sont des *commentaires*. Elles n'ont pas d'influence sur le déroulement du programme. Les (bons) programmeurs mettent des commentaires pour clarifier leur code, ce qui est crucial lorsqu'on travaille en équipe.

La première ligne est une directive `#include < stdio.h >` qui permet d'utiliser les fonctions de la bibliothèque `stdio.h` dans le programme. Cette bibliothèque contient notamment les fonctions d'affichage à l'écran `printf` et de lecture au clavier `scanf`.

Vient ensuite la ligne déclarant le début de la fonction `main`, le programme principal. Le programme principal est la suite des instructions qui seront exécutées. En l'occurrence, il n'y a qu'une seule instruction : un affichage à l'écran par `printf`. Le `\n` permet de passer à la ligne après l'affichage du mot « Bonjour ».

2.3 Lire un nombre

Voici un programme permettant à l'utilisateur de taper un nombre au clavier. Ce nombre est lu par le programme et mémorisé dans une variable `x` qui est un nombre réel (type de données `float`). La variable `x` est ensuite ré-affichée par `printf`.

```
#include <stdio.h>          /* pour pouvoir lire et écrire */

int main(void)              /* programme principal */
{
   float x; /* déclaration d'une variable x (nombre réel) */

   printf("Veuillez entrer un nombre réel au clavier\n");
   scanf("%f", &x); /* lecture au clavier de la valeur de x */
   /* affichage de x : */
   printf("Vous avez tapé %f, félicitations !", x);
   return 0;
}
```

Le format d'affichage et de lecture `%f` correspond à des nombres réels (type `float`). Dans `printf`, lors de l'affichage, le `%f` est remplacé par la valeur de *x*. Par exemple, si l'utilsateur a entré la valeur `15.6` au clavier, le programme affiche la phrase suivante :

```
Vous avez tapé 15.600000, félicitations !
```

Ne pas oubier le & dans le `scanf` ! Cela provoquerait une erreur mémoire (ou *erreur de segmentation*) lors de l'exécution du programme et le programme serait brutalement interrompu.

2.4 Effectuer un calcul et mémoriser le résultat

Le programme suivant mémorise le double de x dans une variable y, par le biais d'une *affectation*. L'affectation (symbole =) permet de stocker le résultat d'un calcul dans une variable.

```
#include <stdio.h>          /* pour pouvoir lire et écrire */

int main(void)              /* programme principal */
{
   float x, y;    /* déclaration de deux variables x et y */

   printf("Veuillez entrer un nombre réel au clavier\n");
   scanf("%f", &x); /* lecture au clavier de la valeur de x */
   y = 2*x; /* on met dans y le double du contenu de x */
   printf("Le double du nombre tapé vaut %f \n", y);
   return 0;
}
```

Le symbole = de l'affectation a une toute autre signification que l'égalité mathématique. L'affectation signifie qu'une variable prend la valeur du résultat d'un calcul. Il correspond à une opération de recopie d'une donnée.

Compléments

√ La syntaxe doit être respectée rigoureusement. Si on oublie un point-virgule, ou bien si on remplace par exemple un guillemet " par une quote ', cela provoque en général une erreur à la compilation. Dans certains cas très précis, il y a plusieurs possibilités pour la syntaxe. Par exemple, le mot `void` dans la déclaration du `main` est facultatif.

√ Parfois, surtout en phase de conception, un programme peut être syntaxiquement correct, mais le comportement du programme n'est pas celui souhaité par le programmeur, en raison d'une erreur de conception ou d'inattention. On dit que le programme comporte un *bogue* (en anglais *bug*).

© Dunod. Toute reproduction non autorisée est un délit.

√ Le `return 0` à la fin du `main` indique seulement que la valeur 0 est retournée au système (ce qui indique que le programme se termine sans erreur). Nous n'utiliserons pas la possibilité de retourner des valeurs au système. Ces fonctionalités sont généralement étudiées avec les systèmes d'exploitation.

Retenir l'essentiel

Un programme, écrit dans un langage informatique, permet de lire des données en entrée, de traiter ces données en appliquant des algorithmes et d'écrire des résultats en sortie. La bibliothèque d'entrées-sorties en *C* s'appelle `stdio.h`. Elle définit des fonctions comme `scanf` et `printf`. Les données sont stockées dans des *variables*, chaque variable ayant son type. Les résultats des calculs peuvent à leur tour être stockés dans des variables au moyen d'une *affectation* (symbole =).

Exercices

2.1 (∗) Pour convertir des degrés Fahrenheit en degrés Celsius, on a la formule suivante :

$$C \simeq 0.55556 \times (F - 32)$$

où F est une température en degrés Fahrenheit et C la température correspondante en degrés Celsius.

a) Écrire un programme *C* qui convertit une température entrée au clavier exprimée en degrés Fahrenheit et affiche une valeur approchée de la même température en degrés Celsius. Les températures seront exprimées par des nombres réels.

b) Même question qu'au a) pour la conversion inverse : de degrés Celsius en degrés Fahrenheit.

2.2 (∗) Lors d'une opération de promotion, un magasin de composants *hardware* applique une réduction de 10% sur tous les composants. Écrire un programme qui lit le prix d'un composant au clavier et affiche le prix calculé en tenant compte de la réduction.

2.3 (∗∗) Soit la fonction mathématique f définie par $f(x) = (2x + 3)(3x^2 + 2)$

a) Écrire un programme *C* qui calcule l'image par f d'un nombre saisi au clavier.

b) Une approximation de la dérivée f' de la fonction f est donnée en chaque point x, pour h assez petit (proche de 0), par :

$$f'(x) \simeq \frac{f(x+h) - f(x)}{h}.$$

Écrire un programme *C* qui calcule et affiche une approximation de la dérivée de f en un point x entré au clavier. On pourra faire saisir le paramètre h au clavier.

2.4 (*c-∗*) Une bille de plomb est lâchée du haut d'un immeuble et tombe en chute libre. Au bout d'un temps t (exprimé en secondes), la bille est descendue d'une hauteur (en mètres) :

$$h = \frac{1}{2} g.t^2$$

avec

$$g = 9.81 \text{ (exprimé en } (m.s^{-2}))$$

a) Écrire un programme qui calcule la hauteur descendue au bout d'un temps t saisi au clavier.

b) Écrire un programme qui calcule la durée totale de la chute connaissant la hauteur totale h de l'immeuble saisi au clavier. (On pourra utiliser la fonction `sqrt` de la bibliothèque `math.h` qui calcule la racine carrée d'un nombre.)

Corrigés

© Dunod. Toute reproduction non autorisée est un délit.

2.1

a)

```
int main(void)
{
  float celsius, fahrenheit;
  printf("Entrez une température en degrés Fahrenheit : ");
  scanf("%f", &fahrenheit);
  celsius = 0.55556 * (fahrenheit - 32.0);
  printf("Température de %f degré Celsius.\n", celsius);
  return 0;
}
```

b)

```
int main(void)
{
  float celsius, fahrenheit;
  printf("Entrez une température en degrés Celsius : ");
  scanf("%f", &celsius);
  fahrenheit = (celsius / 0.55556) + 32.0;
  printf("Température de %f degré Fahrenheit.\n", fahrenheit);
  return 0;
}
```

2.2

```
int main(void)
{
  float prix, prixRemise;
  printf("Entrez un prix : ");
  scanf("%f", &prix);
  prixRemise = 0.9 * prix;
  printf("Le prix avec 10 %% de remise est de %f.\n", prixRemise);
  return 0;
}
```

2.3

a)

```
int main(void)
{
  float x, fx;
  printf("Entrez un nombre : ");
  scanf("%f", &x);
  fx = (2.0 * x + 3.0) / (3.0 * x * x + 2.0);
  printf("f(%f) = %f\n", x, fx);
  return 0;
}
```

b)

```
int main(void)
{
  float x, h, fx, fx_plus_h, fPrime_x;
  printf("Entrez un nombre : ");
  scanf("%f", &x);
  printf("Entrez un écart h : ");
  scanf("%f", &h);
  fx = (2.0 * x + 3.0) / (3.0 * x * x + 2.0);
  fx_plus_h = (2.0 * (x + h) + 3.0) / (3.0 * (x + h) * (x + h) + 2.0);
  fPrime_x = (fx_plus_h - fx) / h;
  printf("f'(%f) = %f\n", x, fPrime_x);
  return 0;
}
```

2.4

a)

```
int main(void)
{
  float h, t;  %

  printf("Entrez une durée : ");
  scanf("%f", &t);
  h = (9.81 * t * t) / 2.0;
  printf("A t = %f, h = %f\n", t, h);
  return 0;
}
```

b) Ne pas oubliez d'ajouter `-lm` à la compilation

```
int main(void)
{
  float h, t;
  printf("Entrez la hauteur totale (en mètres) : ");
  scanf("%f", &h);
  t = sqrt(2.0 * h / 9.81);
  printf("La bille touche le sol au bout de %f secondes\n", t);
  return 0;
}
```

© Dunod. Toute reproduction non autorisée est un délit.