

C le minimum

Prof d'info

Lycée Thiers

À propos

Ces transparents donnent le minimum nécessaire pour commencer à programmer en C.

- écrire et compiler un programme simple ;

À propos

Ces transparents donnent le minimum nécessaire pour commencer à programmer en C.

- écrire et compiler un programme simple ;
- déclarer et utiliser des variables ;

À propos

Ces transparents donnent le minimum nécessaire pour commencer à programmer en C.

- écrire et compiler un programme simple ;
- déclarer et utiliser des variables ;
- afficher des valeurs avec `printf` ;

À propos

Ces transparents donnent le minimum nécessaire pour commencer à programmer en C.

- écrire et compiler un programme simple ;
- déclarer et utiliser des variables ;
- afficher des valeurs avec `printf` ;
- écrire une instruction conditionnelle ;

À propos

Ces transparents donnent le minimum nécessaire pour commencer à programmer en C.

- écrire et compiler un programme simple ;
- déclarer et utiliser des variables ;
- afficher des valeurs avec `printf` ;
- écrire une instruction conditionnelle ;
- écrire une boucle `while` ;

À propos

Ces transparents donnent le minimum nécessaire pour commencer à programmer en C.

- écrire et compiler un programme simple ;
- déclarer et utiliser des variables ;
- afficher des valeurs avec `printf` ;
- écrire une instruction conditionnelle ;
- écrire une boucle `while` ;
- manipuler un tableau statique à une dimension.

Hello world

```
1 #include <stdio.h>
2
3 int main(void){
4     printf("Hello world!\n");
5     return 0;
6 }
```

- `main` est le point d'entrée du programme.

Hello world

```
1 #include <stdio.h>
2
3 int main(void){
4     printf("Hello world!\n");
5     return 0;
6 }
```

- `main` est le point d'entrée du programme.
- Les instructions se terminent en général par un point-virgule.

Hello world

```
1 #include <stdio.h>
2
3 int main(void){
4     printf("Hello world!\n");
5     return 0;
6 }
```

- `main` est le point d'entrée du programme.
- Les instructions se terminent en général par un point-virgule.
- Les accolades délimitent un bloc d'instructions.

Hello world

```
1 #include <stdio.h>
2
3 int main(void){
4     printf("Hello world!\n");
5     return 0;
6 }
```

- `main` est le point d'entrée du programme.
- Les instructions se terminent en général par un point-virgule.
- Les accolades délimitent un bloc d'instructions.
- `printf` permet ici d'afficher une chaîne de caractères.

Compiler et exécuter

Si le programme précédent est enregistré dans `hello.c` :

```
$ gcc -Wall hello.c -o hello
$ ./hello
Hello world!
```

- `gcc` compile le fichier source `hello.c`.

Compiler et exécuter

Si le programme précédent est enregistré dans `hello.c` :

```
$ gcc -Wall hello.c -o hello
$ ./hello
Hello world!
```

- `gcc` compile le fichier source `hello.c`.
- `-Wall` demande l'affichage de nombreuses mises en garde.

Compiler et exécuter

Si le programme précédent est enregistré dans `hello.c` :

```
$ gcc -Wall hello.c -o hello
$ ./hello
Hello world!
```

- `gcc` compile le fichier source `hello.c`.
- `-Wall` demande l'affichage de nombreuses mises en garde.
- `-o hello` donne le nom `hello` au fichier exécutable.

Compiler et exécuter

Si le programme précédent est enregistré dans `hello.c` :

```
$ gcc -Wall hello.c -o hello
$ ./hello
Hello world!
```

- `gcc` compile le fichier source `hello.c`.
- `-Wall` demande l'affichage de nombreuses mises en garde.
- `-o hello` donne le nom `hello` au fichier exécutable.
- `./hello` exécute ce fichier.

Avant la compilation : le préprocesseur

Les lignes commençant par `#` sont traitées avant la compilation proprement dite.

```
1 #include <stdio.h>
```

- `stdio.h` est un fichier d'en-tête de la bibliothèque standard.

Avant la compilation : le préprocesseur

Les lignes commençant par `#` sont traitées avant la compilation proprement dite.

```
1 #include <stdio.h>
```

- `stdio.h` est un fichier d'en-tête de la bibliothèque standard.
- Il contient notamment les déclarations nécessaires pour utiliser `printf`.

Avant la compilation : le préprocesseur

Les lignes commençant par `#` sont traitées avant la compilation proprement dite.

```
1 #include <stdio.h>
```

- `stdio.h` est un fichier d'en-tête de la bibliothèque standard.
- Il contient notamment les déclarations nécessaires pour utiliser `printf`.
- L'étape qui traite les directives comme `#include` s'appelle le *prétraitement* (ou preprocessing).

Variables et affichage

```
1 #include <stdio.h>
2
3 int main(void){
4     int n = 12;
5     double x = 3.14;
6
7     printf("n = %d\n", n);
8     printf("x = %f\n", x);
9     printf("bonjour\n");
10    return 0;
11 }
```

- Une variable est déclarée en donnant son type puis son nom.

Variables et affichage

```
1 #include <stdio.h>
2
3 int main(void){
4     int n = 12;
5     double x = 3.14;
6
7     printf("n = %d\n", n);
8     printf("x = %f\n", x);
9     printf("bonjour\n");
10    return 0;
11 }
```

- Une variable est déclarée en donnant son type puis son nom.
- = permet d'affecter une valeur.

Variables et affichage

```
1 #include <stdio.h>
2
3 int main(void){
4     int n = 12;
5     double x = 3.14;
6
7     printf("n = %d\n", n);
8     printf("x = %f\n", x);
9     printf("bonjour\n");
10    return 0;
11 }
```

- Une variable est déclarée en donnant son type puis son nom.
- = permet d'affecter une valeur.
- `%d` affiche un `int`, `%f` un `double`.

Variables et affichage

```
1 #include <stdio.h>
2
3 int main(void){
4     int n = 12;
5     double x = 3.14;
6
7     printf("n = %d\n", n);
8     printf("x = %f\n", x);
9     printf("bonjour\n");
10    return 0;
11 }
```

- Une variable est déclarée en donnant son type puis son nom.
- = permet d'affecter une valeur.
- `%d` affiche un `int`, `%f` un `double`.
- `\n` provoque un passage à la ligne.

Instruction conditionnelle

```
1 int n = 12;
2
3 if (n % 2 == 0){
4     printf("pair\n");
5 }
6 else{
7     printf("impair\n");
8 }
```

- La condition est placée entre parenthèses.

Instruction conditionnelle

```
1 int n = 12;
2
3 if (n % 2 == 0){
4     printf("pair\n");
5 }
6 else{
7     printf("impair\n");
8 }
```

- La condition est placée entre parenthèses.
- Les blocs sont délimités par des accolades.

Instruction conditionnelle

```
1 int n = 12;
2
3 if (n % 2 == 0){
4     printf("pair\n");
5 }
6 else{
7     printf("impair\n");
8 }
```

- La condition est placée entre parenthèses.
- Les blocs sont délimités par des accolades.
- == teste l'égalité ; = est une affectation.

Instruction conditionnelle

```
1 int n = 12;
2
3 if (n % 2 == 0){
4     printf("pair\n");
5 }
6 else{
7     printf("impair\n");
8 }
```

- La condition est placée entre parenthèses.
- Les blocs sont délimités par des accolades.
- == teste l'égalité ; = est une affectation.
- Comparaisons usuelles : ==, !=, <, <=, >, >=.

Boucle `while`

```
1 int i = 0; // déclaration du compteur
2
3 while (i < 5){
4     printf("%d\n", i); // afficher avec passage à la ligne
5     i = i + 1; // incrémenter le compteur
6 }
```

- Tant que la condition est vraie, le bloc est répété.

Boucle `while`

```
1 int i = 0; // déclaration du compteur
2
3 while (i < 5){
4     printf("%d\n", i); // afficher avec passage à la ligne
5     i = i + 1; // incrémenter le compteur
6 }
```

- Tant que la condition est vraie, le bloc est répété.
- Ici la variable `i` sert de compteur.

Boucle `while`

```
1 int i = 0; // déclaration du compteur
2
3 while (i < 5){
4     printf("%d\n", i); // afficher avec passage à la ligne
5     i = i + 1; // incrémenter le compteur
6 }
```

- Tant que la condition est vraie, le bloc est répété.
- Ici la variable `i` sert de compteur.
- Il faut veiller à ce que la condition puisse finir par devenir fausse, sinon la boucle est infinie.

Boucle `while`

```
1 int i = 0; // déclaration du compteur
2
3 while (i < 5){
4     printf("%d\n", i); // afficher avec passage à la ligne
5     i = i + 1; // incrémenter le compteur
6 }
```

- Tant que la condition est vraie, le bloc est répété.
- Ici la variable `i` sert de compteur.
- Il faut veiller à ce que la condition puisse finir par devenir fausse, sinon la boucle est infinie.
- En Français on compte à partir de 1, en info à partir de 0

Tableaux statiques à une dimension

```
1 int t[5] = {4, 7, 2, 9, 1}; //déclaration tableau 5 items
2
3 printf("%d\n", t[0]); //afficher l'élément numéro 0
4
5 t[2] = 12; //modifier l'élément numéro 2
6
7 int i = 0; //initialisation du compteur
8 while (i < 5){
9     printf("%d\n", t[i]); // afficher l'élément numéro i
10    i = i + 1;
11 }
```

- `int t[5]` déclare un tableau de 5 entiers.

Tableaux statiques à une dimension

```
1 int t[5] = {4, 7, 2, 9, 1}; //déclaration tableau 5 items
2
3 printf("%d\n", t[0]); //afficher l'élément numéro 0
4
5 t[2] = 12; //modifier l'élément numéro 2
6
7 int i = 0; //initialisation du compteur
8 while (i < 5){
9     printf("%d\n", t[i]); // afficher l'élément numéro i
10    i = i + 1;
11 }
```

- `int t[5]` déclare un tableau de 5 entiers.
- Les indices vont de 0 à 4.

Tableaux statiques à une dimension

```
1 int t[5] = {4, 7, 2, 9, 1}; //déclaration tableau 5 items
2
3 printf("%d\n", t[0]); //afficher l'élément numéro 0
4
5 t[2] = 12; //modifier l'élément numéro 2
6
7 int i = 0; //initialisation du compteur
8 while (i < 5){
9     printf("%d\n", t[i]); // afficher l'élément numéro i
10    i = i + 1;
11 }
```

- `int t[5]` déclare un tableau de 5 entiers.
- Les indices vont de 0 à 4.
- `t[i]` permet de lire ou de modifier l'élément d'indice `i`.