Une introduction au C/C++ non orienté objet

Un premier programme en C/C++ : comme il est classique, ce programme écrit bonjour à l'écran.

Les types de donnée simples : int, float, double, char, les tableaux, les tableaux alloués dynamiquement.

Les structures de contrôle : les tests et les boucles

Les fonctions et sous-programmes : un sous-programme est une fonction de type void en C/C++

Les exceptions

La généricité pour les fonctions

Les entrées sorties : on présente ici à la fois les entrées sorties en C et en C++ (qui sont en fait orientées objet). Les entrées sorties de style C sont utilisables en C++. On trouvera les entrées sorties à l'écran, dans un fichier ou une chaîne de caractères.

Un premier programme en C++ (Version C++ standard)

```
#include <iostream>
// iostream sans .h !!
int main()
{
std::cout << "Bonjour\n"; // on écrit à l'écran en version C++ standard
return(0);
}
```

ou

```
#include <iostream>
// iostream sans .h !!
using namespace std;
// cout se trouve dans l'espace de nom std
int main()
{
cout << "Bonjour\n"; // on écrit à l'écran en version C++ standard
return(0);
}
```

Note

- Les instructions se terminent par un point-virgule ;
- Le commentaire part de // et va jusqu'à la fin de la ligne.
- On peut utiliser une [syntaxe purement C pour les commentaires](#) .
- Vous trouverez ici un [exemple C++ de lecture à l'écran](#)
- Pour plus de détails sur les entrées sorties, [voir la partie du cours consacrée à cette question](#) .
- Pour des raisons de compatibilité, [la version C++ pré-standard \(avec iostream.h\)](#) sera reconnue par les compilateurs récents. Les anciens compilateurs, quant à eux, ne comprendront rien à la syntaxe C++ standard.

Vieille syntaxe des commentaires en C

Version C

```
#include <stdio>
int main()
{
printf( "Bonjour\n"); /* Ceci est un commentaire
ancienne maniere */
return(0);
}
```

Notes

- le commentaire se trouve entre /* et */, ce qui permet de commenter facilement plusieurs lignes, mais également d'avoir des messages d'erreur incompréhensibles quand par inadvertance on tente de commenter une zone contenant déjà un commentaire.
- Il est donc conseillé d'utiliser [la syntaxe C++ pour les commentaires](#) .

Un exemple de lecture a l'ecran en C++

Version C++

```
#include <iostream>
using namespace std;
int main()
{
float a,b;
cout << "donnez la valeur de a et b" << endl ;
// on ecrit a l'ecran en version C++
cin >> a >> b;           // on lit a l'ecran en version C++
cout << "a= " << a << " b= " << b << endl;
return(0);
}
```

Notes

Faire attention au sens : cout << et cin >>

endl est à peu pres la meme chose que \n, sauf que l'on est sûr que la ligne va etre imprimée immediatement.

Pour plus de details sur les entrées sorties, [voir la partie du cours consacrée a cette question](#) .

Types de données simples

- Le type entier et quelques fonctions associées: int .
 - les types reels float et double et les fonctions mathematiques.
 - Le type caractère char et les chaines de caractères.
 - Les tableaux
- Dans tous les cas, l' affectation utilise le signe = en C/C++**
- en C++, on passe d'un type de donnée a un autre de la maniere suivante:
int a,c; double b; b=double(a)/double(c) ;
L'equivalent en C est plus ou moins inavouable.
- Il y a aussi la possibilite d'une conversion automatique
double a,b; b=2; a=b+1; // 1 est un entier contrairement a 1.0, qui est un double par defaut

Le type entier int

```
int a; int b,c; int d=5;
```

Opérations sur des entiers

```
a=b+1; a=b-2; a=b*2;  
a=3/2;    // attention : division entiere, le resultat vaut 1  
a=7%3;    // 7 modulo 3 donne 1
```

Incrémentation et décrémentation

```
a++;    // incrémente a de 1  
b--;    // décrémente b de 1
```

Distinguer ++a et a++

```
b=a++;    // équivalent a    b=a; a=a+1;  
b=++a;    // équivalent a    a=a+1;b=a;  
  
b+=2;     // équivalent a    b=b+2;  
b-=2;     // équivalent a    b=b-2;  
b*=2;     // équivalent a    b=b*2;  
b/=2;     // équivalent a    b=b/2;
```

Tous ces operateurs s'appliquent a des nombres reels de type float et double mais sont moins utilisés dans ce cas.

Exemple

```
#include <iostream>  
using namespace std;  
int main()  
{  
int a,b;  
cout << "donnez la valeur de a et b" << endl ;  
cin >> a >> b;  
a++; b-=2;           // on lit a l'ecran en version C++  
int c=5; c--;  
// on peut (en C++ uniquement) definir et initialiser des variables en plein milieu d'une procedure  
// Attention, ne pas les redéfinir par la suite  
cout << a << " " << b << " " << c << "\n" ;  
return(0);  
}
```

Les types réels float et double

float : réel en simple precision

double: réel en double precision (utile pour le calcul numérique)

fonctions usuelles + - * /

fonctions mathématiques : exemple

```
#include <iostream>
#include <cmath> // equivalent de #include <math.h> en C
using namespace std;
int main()
{
float a,b;
a=1; b=0;
cout << exp(a) <<" " << cos(b) << endl;
return(0);
}
```

La librairie mathématique contient les fonctions usuelles

fabs, sin, cos, tan, atan, cosh, sinh, log, exp, sqrt, pow (pow(x,y) x puissance y) ...

A signaler quelques défauts : les fonctions min, max, sqr ne sont pas définies

(les définir inline)

min

```
inline double min( double a, double b)
{
return((a<b) ? a :b);
}
```

max

```
inline double max( double a, double b)
{
return((a>b) ? a :b);
}
```

sqr

```
inline double sqr( double a)
{
return(a*a);
}
```

Les fonctions min, max et sqr, qui manquent cruellement à la librairie mathématique du C/C++

peuvent être également définies avec des arguments de type float et int .

Le type char et les chaînes de caractères

```
#include <iostream>
```

```
int main()
{
char a,b,c;
a='l'; b='z';c='r';
std::cout << a << b << c << std::endl;
std::cout << "bonjour\n";
return(0);
}
```

"bonjour" est une chaîne de caractères constante composée des caractères
'b' 'o' 'n' 'j' 'o' 'u' 'r' '\0'
'c' et "c" ne sont donc pas la même chose.

Les tableaux en C/C++

Exemple

```
#include <iostream>
int main()
{
float a[10];
for (int i=0; i <10;i++)
{ a[i]=float(i); }
return(0);
}
```

Note 1

Les tableaux en C/C++ (Java) commencent obligatoirement a 0.

Commentaire: Cette restriction complique inutilement la programmation d'algorithmes numeriques en C/C++

Note 2

Contrairement a certains autres langages, les tableaux ne sont jamais passes par valeur en C/C++

Exemple:

```
#define m 20
void initialise(float a[], int n)
// void initialise(float a[m], int n) est egalement possible
{
for (int i=0; i <n; i++) a[i]=1.0;
}
int main()
{
float a[m];
initialise(a,m);
return(0);
}
```

Il est possible d'allouer dynamiquement de la memoire à un tableau

Les tableaux multidimensionnels

```
float a[10][20];
for (int i=0; i <10; i++)
{
    for (int j=0; j <20; j++)
    {
        a[i][j]=1.0;
    }
}
```

Passage d'un tableau multidimensionnel en argument d'une fonction

```
void initialise(float a[][20])
// void initialise(float a[10][20]) est également possible.
// Obligatoirement specifier la deuxième dimension
{
for (int i=0; i <10; i++)
{
    for (int j=0; j <20; j++)
    {
        a[i][j]=0.0;
    }
}
}
int main()
{
float a[10][20]; initialise(a);
return 0;
}
```

Dans tous les cas, il n'est pas possible de passer les dimensions du tableau en argument de la fonction. On utilise donc typiquement le préprocesseur:

```
#define n1 10
#define n2 20
void initialise(float a[n1][n2] )
{
// ...
}
```

L'allocation dynamique de mémoire

En C/C++, on peut allouer (et libérer) dynamiquement la mémoire. Nous utiliserons ici une syntaxe typiquement C++, faisant appel aux opérateurs *new* et *delete*.

Remarque importante : La libération de mémoire peut conduire à des bugs redoutables quand on cherche à réutiliser par exemple un tableau dont on a libéré la mémoire. Pour éviter ces problèmes, certains langages choisissent de refuser au programmeur le droit de libérer lui même la mémoire, et demandent à un processus appelé *garbage collector* de réaliser cette désallocation. Cette possibilité n'existe pas en C++, aussi, à un niveau élémentaire, on sera très prudent lorsqu'on libère la mémoire.

L' allocation de mémoire pour des tableaux

Exemple:

```
#include <iostream>
int main()
{
int n;
double *a;

// a est un pointeur sur un double
// il va suffire d'allouer la bonne quantité de mémoire pour que cela devienne
// un pointeur sur le premier élément du tableau
std::cout << "rentrez n \n"; std::cin >>n;

a= new double[n]; // allocation d'un tableau de taille n

// attention , les éléments du tableau n'ont aucune raison d'être initialisés à 0
for (int i=0; i<n;i++) { a[i]=1.0; std::cout << a[i] << "\n" ; } // le tableau s'utilise normalement une
fois alloué

// Si l'on voulait libérer l'espace mémoire alloué au tableau
// on écrirait simplement
// delete []a;

// bien entendu tenter de libérer 2 fois l'espace mémoire associé à a
//. aurait toutes les raisons de provoquer de sérieux problèmes

return 0;
}
```

La mémoire allouée dans n'importe quelle fonction du programme reste allouée partout ailleurs dans le programme.

[L' allocation de mémoire pour les objets](#)

Structures de contrôle

- Les tests

- Les boucles

Les tests

La structure if

```
int i;
if (0==i)
{
// placer votre code ici
}
else if (1==i)
{
// placer votre code ici
}
else
{
// placer votre code ici
}
```

Note: les else if et else sont bien entendu facultatifs

ATTENTION: IL FAUT == POUR LA COMPARAISON

ne pas écrire `if (i=1)`
: ça se compile, ça s'exécute, et ça fait n'importe quoi .
Vous êtes donc priés d'écrire plutôt
if (1==i) qui déclenche une erreur de compilation si on oublie un = .

egalite == , différent != , < , <= , > , >=

```
if (1== i && 0==j) a++; // && et logique
if (1== i || 0==j) a++; // || ou logique
if (i) a++; // test vrai si i n'est pas egal a 0
if (!i) a++; // ! non logique
```

La structure switch

```
#include <iostream>
int main()
{
char reply;
std::cout << "voulez vous continuer ?";
std::cin >> reply;
switch (reply)
{
case 'O' :
case 'o':
// placer le code pour oui
break;
case 'N' :
case 'n':
// placer le code pour non
break;
default:
// placer le code pour tout autre reponse
break;
}
}
```

Les différents types de boucles

La boucle for

```
int i;
for (i=0;i<33;i++)
{
// placer votre code ici
}
```

La boucle while

```
int i=0;
while (i<33)
{
// placer votre code ici
i++;
}
```

La boucle do

```
int i=35;
do
{
// placer votre code ici
i++;
} while (i<33);
```

La boucle do, toujours exécutée au moins une fois, est beaucoup moins utile que les boucles for et while .

La boucle infinie

```
char reply;
for (;;)
{
cout << " voulez vous arreter la boucle";
cin >> reply;
if ( 'o' == reply) break;
}
```

Quelques possibilités évoluées

```
for (int i=0; i<12; i++)  
{  
if ( 10 == i) continue;  
cout <<i <<"\n";  
}
```

```
int i, j;  
for (i=0,j=0; i <10; i++,j--) std::cout << i <<" " <<j << std::endl;
```

Les fonctions

Avant toute chose, par défaut en C/C++ , les arguments sont passés par valeur (sauf les tableaux).

Egalement : une fonction ne peut pas être définie à l'intérieur d'une autre fonction

[voir ici les passages par référence et par pointeur](#)

[voir ici les passages de fonctions en argument d'autres fonctions](#)

Exemple de définition de fonction

```
double f0(double x, double y)
{
double z;
z=x*x+y;
return z;
}
```

appel par

```
double a,b,c;
a=f0(b,c);
```

Une fonction n'a pas forcément d'argument

```
int un()
{
cout << "fonction un";
return 1;
}
```

appel par

```
int i0;
i0=un();
```

Une fonction peut être de type void (vide: une procédure en fait)

```
void imprime( char c)
{
cout << c <<endl ;
return; // le return est facultatif dans ce cas
}
```

appel par

```
char c0;
imprime(c0);
```

Les fonctions sont déclarées avant le programme principal

```
#include <iostream>
using namespace std;
float x2=6.0;          // variable globale
float cube(float x)
{
return(x*x*x);
}
int main()
{
float x0,y;
x0=x2;
y=cube(x0);
cout << y << endl;
return 0;
}
```

On peut aussi n'écrire que l'interface (le prototype) de la fonction avant le programme principal

(l'implémentation étant écrite a un autre endroit)

```
#include <iostream>
using namespace std;
float x2=6.0;          // variable globale
float cube(float x);  // attention au point virgule dans ce cas
int main()
{
float x0,y;
x0=x2;
y=cube(x0);
cout << y << endl;
return 0;
}

float cube(float x)
{
return(x*x*x);
}
```

Les interfaces et implémentations des différentes fonctions se trouveront dans des fichiers différents pour de gros projets. Dans l'exemple précédent , la ligne

```
float cube(float x);
```

se trouverait dans le fichier cube.h, l'implémentation dans le fichier cube.cpp, et on aurait au début la ligne

```
#include "cube.h"
```

Les fonctions peuvent être récursives

```
void recursive(int n)
{
int profondmax=10;
if (n== profondmax) return;
else
{
// inserez votre code ici
recursive(n+1);
}
return;
}
```

appel par

```
recursive(1);
```

En C++, une fonction peut avoir des arguments par défaut

```
float ajoute( float x=1.0, float y=2.0)
{
return x+y;
}
```

appel par

```
float z;
z=ajoute(3,4); // 3+4
z=ajoute(3); // 3 + 2 (valeur par default)
z=ajoute(); // 1 + 2 ( les 2 arguments prennent leur valeur par default)
// Attention, on ne peut pas donner au 1er argument sa valeur par default
// tout en modifiant le 2ieme
```

En C++, une fonction peut être inline

On rajoute inline devant la fonction, la fonction est remplacée automatiquement par son contenu partout dans le programme.

```
inline float sqr(float x)
{
return x*x;
}
```

Les variables sont réinitialisées a chaque appel, **sauf si on les déclare static**

```
#include <iostream>
using namespace std;
double incremente( double x)
{
static double total = 0 ; // initialisé seulement au debut
total += x;
return total;
}
int main()
{
double y;
y = incremente(10);
cout << y << endl;
y = incremente(20);
cout << y <<endl;
return 0;
}
```

Des fonctions similaires opérant avec des arguments de type différents ou un nombre différent d'arguments peuvent avoir le même nom, on appelle cela surcharge .

Les passages d'argument par référence et par pointeur

Par défaut les arguments sont passés par valeur (sauf les tableaux).

Pour modifier un argument, on peut le passer par référence (en C++)

```
void echange(float &x, float &y)    // float &x -> passage de x par reference ici
{
float temp;
temp = x;
x = y;
y = temp;
}
```

appel par

```
float a,b;
echange(a,b);
```

Pour modifier un argument, on peut aussi le passer par pointeur (en C/C++)

```
void echange(float *x, float *y)    // *x pointeur sur x
{
float temp;
temp = *x;
*x = *y;
*y = temp;
}
```

appel par

```
float a,b;
echange(&a,&b);                // &a represente ici l'adresse de a
```

Les passages de fonctions en argument

On utilise pour cela des pointeurs sur des fonctions

Exemple:

```
#include <iostream>
double f( double x)
{
return exp(x);
}

double g( double x)
{
return cos(x);
}

double compose(double (*a)( double x), double (*b)( double y), double z)
// x et y facultatifs
// attention ne pas oublier les parentheses dans double (*a)()
// sinon au lieu d'un pointeur sur une fonction retournant un double
// ce serait une fonction retournant un pointeur sur un double
{
return a(b(z));
// pas besoin de * ici
}

int main()
{
double z=0,w;
w=compose(f,g,z);
// le nom de la fonction donne l'adresse de la fonction (pas de &)
std::cout <<w << "\n";
return 0;
}
```

La surcharge des fonctions (C++)

Des fonctions différentes peuvent porter le même nom , elles se distingueront par le nombre et le type de leurs arguments. On parle de **surcharge**.

On veillera a n'appeler par le même nom que des fonctions réalisant des operations similaires.

Exemple 1

```
double ajoute(double x, double y)
{
return x+y;
}
```

```
int ajoute(int x, int y)
{
return x+y;
}
```

```
double a,b,c; int i,j,k;
a=ajoute(b,c); // appelle la fonction opérant sur des double
i=ajoute(j,k); // appelle la fonction opérant sur des entiers
```

Exemple 2

```
#include<cmath>          // mettre ca au debut
double norme(double x)
{
return fabs(x);
}
double norme(double x, double y)
{
return sqrt(x*x+y*y);
}
double norme(double x, double y, double z)
{
return sqrt(x*x+y*y+z*z);
}

double a,b,c,d;
a=norme(b);           // appelle la fonction à un argument
a=norme(b,c);        // appelle la fonction à deux arguments
a=norme(b,c,d);      // appelle la fonction à trois arguments
```

[Voir ici le polymorphisme pour les objets](#)

Les exceptions en C++

Les exceptions, introduites au départ dans le langage ADA, sont un moyen très efficace de traiter les erreurs. Il ne faut cependant pas en abuser car :

- Elles peuvent conduire à une exécution lente.
- Mal utilisées , elles peuvent donner des bugs difficiles à détecter.

Exemple:

Dans l'exemple suivant, nous allons voir les possibilités des exceptions qui vont nous permettre ici de sortir en catastrophe d'une boucle et de deux fonctions:

```
#include <iostream>
using namespace std;
enum exception2 {Erreur1,Erreur2};
void procedure2() throw (exception2)
{
int n=20,catastrophe1=0,catastrophe2=0;
for(int i=0;i<n;i++)
{
if (catastrophe1) throw Erreur1;
if (catastrophe2) throw Erreur2;
if (10==i) catastrophe1=1;
cout <<i <<endl;
}
return;
}
void procedure1() { procedure2(); return; }
int main()
{
try
{
procedure1();
}
catch (exception2 e)
{
if (e==Erreur1) cout << "Erreur1 a l'interieur de procedure2\n";
if (e==Erreur2) cout << "Erreur2 a l'interieur de procedure2\n";
}
return 0;
}
```

La généricité pour les fonctions

Une fonction générique est paramétrée (mot-clé **template**) par un nom de type prédéfini (int, float ...) ou de classe créée par le programmeur. La fonction générique va donc pouvoir fonctionner avec des arguments de différents types.

Exemple:

```
template <class Type>
Type max2(Type x, Type y) {
// on pourrait même mettre la fonction inline
return (x>y ? x : y);
}
```

l'appel max(a,b) marchera aussi bien sur des int, des float que des double.

Attention, ne pas mélanger des arguments de type différents dans l'appel de la fonction précédente.

On pourrait utiliser deux noms de type :

```
template <class Un, class Deux>
```

[Voir ici la généricité pour les classes](#)

Les entrées sorties en C/C++

Les entrées sorties sont très différentes en version C et C++. Nous présenterons ici à chaque fois les deux versions.

Les entrées sorties à l'écran et le formatage

Les entrées sorties dans un fichier

Les entrées sorties dans une chaîne de caractère (string)

les entrées sorties caractère par caractère ou ligne par ligne

Les entrées sorties à l'écran

Ecrire

Ecrire en C

```
#include <stdio>

int main()
{ int i=2; float f=4.5; char s[]="chaine" ;
printf("nous allons ecrire un entier en decimal %d \n un reel (float ou double) %f \n et un string %s\n",i,f,s);
return 0; }
```

A noter d'autres possibilités:

- réel (entier ou double) en notation scientifique %e
- entier en hexadécimal (base 16) %x
- entier en octal (base 8) %o
- caractère %c

Ainsi que des écritures plus précises:

- %5d entier en décimal et 5 caractères
- %8.5f réel avec 8 caractères en tout (y compris le - éventuel et le .) et 5 chiffres après la virgule.
- %12.5e réel en notation scientifique avec 12 caractères et 5 chiffres après la virgule.

Ecrire en C++

```
#include <iostream>
#include <string>
int main()
{
int i=2; float f=4.5; std::string s="chaine" ; // evidemment on peut mettre using namespace std;
std::cout << "entier " << i << " reel " << f << " string " << s << "\n";
return 0;
}
```

A noter:

- la sortie d'erreur standard (cerr au lieu de cout)
- endl est équivalent à "\n" mais impose en plus d'écrire immédiatement (important en cas de plantage)

Lire

Lire en C

```
#include <stdio>
int main()
{
int i; float f; char s[128];
printf("ecrivez un entier un reel et un string\n");
scanf("%d %f %s",&i,&f,s); // attention aux & !!
printf("%d %f %s\n",i,f,s);
return 0;
}
```

Lire en C++

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
int i; float f; string s ;
cout << "ecrivez un entier un reel et un string\n";
cin >>i >> f >>s;
cout <<i << " " <<f << " " <<s << endl;
return 0;
}
```

Les entrées sorties dans des fichiers

Version C

Ecrire

```
#include <stdio>
int main()
{
float d=9.6,d2=5.4;
FILE *out;
out=fopen("fichier","w");
if (out == NULL) {
printf("out n'a pas ete ouvert\n");
return -1;
}
fprintf(out,"%e\n",d);
fprintf(out,"%e\n",d2);
fclose(out); // fermeture facultative
return 0;
}
```

Lire

```
#include <stdio>
int main()
{
float d,d2;
FILE *in;
in=fopen("fichier","r");
if (in == NULL) {
printf("in n'a pas ete ouvert\n");
return -1;
}
fscanf(in,"%e",&d);
fscanf(in,"%e",&d2);

printf("%e %e\n",d,d2);

fclose(in); // fermeture facultative
return 0;
}
```

Version C++

Ecrire

```
#include <iostream>
#include <fstream>

using namespace std;

int main()
{
float d=9.6,d2=5.4;
ofstream out("fichierc++");
// if (!out) {
// cerr << "out n'a pas ete ouvert\n";
// return -1;
// }
out << d << "\n" << d2 << "\n";
out.close(); // fermeture facultative
return 0;
}
```

Lire

```
#include <iostream>
#include <fstream>

using namespace std;

int main()
{
float d,d2;
ifstream in("fichierc++");
// if (!in) {
// cerr << "in n'a pas ete ouvert\n";
// return -1;
// }
in >> d >> d2 ;
cout << d << " " << d2 << endl;
in.close(); // fermeture facultative
return 0;
}
```

Les entrées sorties dans des chaînes de caractères

Version C

Ecrire

```
#include <stdio>
int main()
{
double d=9.6,d2=5.4;
char chaine[80];
sprintf(chaine,"%e %e",d,d2);
printf("%s\n",chaine);
return 0;
}
```

Version C++

Ecrire et lire

```
#include <iostream>
#include <sstream>
#include <string>
using namespace std;
int main()
{
double d=9.6,d2=5.4;
string chaine;
ostringstream out;
out << d << " " << d2 << endl ;
chaine = out.str();

cout << chaine ;

string s2 = "34 22";
int a,b;
istringstream ins;
ins.str(s2);
ins >> a >> b;
cout << a << " " << b << endl;

return 0;
}
```

Les entrées sorties caractère par caractère

Version C

Ecrire

```
#include <stdio>
int main()
{
FILE *out;
out=fopen("fich0","w");
if (out == NULL) {
printf("out n'a pas ete ouvert\n");
return -1;
}
putc('c',out); // attention, le sens est le contraire de fprintf
putc(33,out); // code ASCII correspondant au caractère
fclose(out); // fermeture facultative
return 0;
}
```

Lire

```
#include <stdio>
int main()
{
int a,b;
FILE *in;
in=fopen("fich0","r");
if (in == NULL) {
printf("in n'a pas ete ouvert\n");
return -1;
}
a=getc(in); b=getc(in);
// on récupère en fait les codes ASCII des caractères
printf("%c %c\n",a,b);
fclose(in); // fermeture facultative
return 0;
}
```

Version C++

Ecrire

```
#include <iostream>
#include <fstream>

using namespace std;

int main()
{
    ofstream out("fich1");
    out.put('f');
    out.close();    // fermeture facultative
    return 0;
}
```

Lire

```
#include <iostream>
#include <fstream>

int main()
{
    char c;
    std::ifstream in("fich1");
    c=in.get() ;
    std::cout << c << "\n";
    in.close();    // fermeture facultative
    return 0;
}
```