Les pointeurs sur fonctions

Par uknow

www.siteduzero.com

Dernière mise à jour le 2/05/2012

Sommaire

Les pointeurs sur fonctions

Par  uknow

Mise à jour : 08/04/2011
Difficulté : Facile Durée d'étude : 1 heure

Votre histoire avec les pointeurs continue , et cette fois-ci c'est de fonctions qu'il s'agit.
La compréhension de ce tutoriel exigera un minimum de connaissances des pointeurs (à ne pas confondre avec les allocations dynamiques).

Bonne lecture.

Sommaire du tutoriel :

Le pointeur générique (void *)

Bonne nouvelle , je vais vous faire un petit rappel sur les pointeurs :

Quand on déclare une variable, une place mémoire lui sera attribuée. Cette place mémoire va servir pour contenir les valeurs qu'on affectera à notre variable. Et cet espace sera connu par une adresse.

Ainsi chaque objet créé en mémoire, a une adresse propre à lui. Cette adresse peut être stockée dans un type qu'on appelle ***pointeur***.

Code : C

```
int * pointeur;
```

Jusqu'ici vous avez connu, plusieurs types de pointeurs (sur `char`, `int`, `float`...) :

Code : C

```
int * ptrint;
float * ptrfloat;
char * ptrchar;
double * ptrdouble;
```

Vous avez également appris que les types de pointeurs n'étaient pas pareil et qu'il ne fallait pas les mélanger (un pointeur sur `int`, doit contenir une adresse d'une variable `int`, par exemple), ce qui est une chose vraie. Cependant, il existe un type de pointeur compatible avec tous les autres types de pointeurs sur objets qu'on appelle le pointeur générique (désigné par `void *`).

Code : C

```
void * pointeurGenerique;
```

Quel est le rapport avec les fonctions ?

Une fonction, comme tout autre objet dans un programme, a également une adresse mémoire et donc un pointeur capable de sauvegarder cette adresse mémoire. Pour qu'elle soit utilisée par la suite. Cependant, cette adresse ne pourrait pas être stockée dans le pointeur générique de façon entièrement portable.

Comment peut-on avoir l'adresse d'une fonction ?

Identiquement aux variables ordinaires que vous connaissez si bien, on peut utiliser l'opérateur '&', suivi du nom de la fonction pour avoir son adresse.

Exemple :

Si j'ai la fonction 'ma_fonction' comme ceci :

Code : C

```
void ma_fonction(void)
{
     //instructions...
}
```

En faisant :

Code : C

```
pointeurSurFonction = &(ma_fonction);
```

Cela stockera l'adresse de notre fonction dans la variable 'pointeurSurFonction'.

Il existe une autre façon plus simple pour avoir l'adresse d'une fonction, j'ai préféré commencer par la plus dure  .

Le nom d'une fonction est un pointeur dit statique dessus (pour les maniaques du langage, j'utiliserai les termes "le nom de fonction est évalué en pointeur dessus").

En conséquence, l'utilisation de l'opérateur '&' est donc facultative, ainsi seulement par l'utilisation du nom de la fonction on récupère son adresse.
Une nouvelle fois vous ne serez pas perdant en utilisant des expressions explicites, et le '&' devant est explicite.

Code : C

```
pointeurSurFonction = ma_fonction;
```

Maintenant nous savons comment déclarer un pointeur générique, et nous savons également lire l'adresse d'une fonction, alors si on combine les deux notions ceci nous donne un code comme ceci :

Code : C

```
void ma_fonction (void)
{
     printf("Hello world!\n");
}

void * pointeurGenerique = ma_fonction ;
```

Ainsi on a gardé l'adresse de notre fonction dans le pointeur générique.
Notez que ce pointeur `void*` ne vous permettra pas d'appeler votre fonction.

Attention :
Comportement non standard! un grand nombre d'extensions de compilateurs supportent qu'on affecte un pointeur sur fonction au type void *, mais il reste **non-portable**.

chouette, et ça nous sert à quoi ?

à rien pour l'instant, c'était juste une petite introduction à notre sujet qui a pour but arriver à appeler une fonction par son adresse.

Avant de passer à la suite, je tiens à vous dire que les pointeurs de fonctions ont aussi des types, liés aux arguments de la fonction (nombre, et types) et aussi à son type de retour.
C'est un détail qui fait toute la différence, et qu'il faudra respecter pour parvenir à appeler notre fonction.

Fonctions sans retour (procédures)

Généralement, un pointeur sur fonction est déclaré suivant la syntaxe :

type_de_retour `(* nom_du_pointeur_sur_la_fonction )` **(liste_des_arguments**);

Ainsi ces pointeurs sont déclarés en respectant le prototype de la fonction sur laquelle ils vont pointer. Ce qui est logique  , non ?

Fonction sans paramètres (sans arguments)

On appelle procédure ou fonction sans retour, toutes fonctions ayant le type `void` comme type de retour.

Exemple :

Code : C

```
void ma_fonction(....);
```

Et on appelle fonction sans arguments (ou sans paramètres), toutes fonctions ayant le type `void` en argument.

Exemple:

Code : C

```
void ma_fonction(void);
```

Ainsi la déclaration d'un pointeur sur ce type de fonctions, à savoir sans arguments et sans retour, est comme ceci :

Code : C

```
void (*pointeurSurFonction)(void);
```

Le void à gauche est le type de retour, et celui entre parenthèses (à droite) est le type d'arguments. On a donc déclaré un pointeur sur fonction du type décrit ci-dessus. Et qui peut être utilisé comme ceci :

Code : C

```
void ma_fonction(void);                 /*Prototype*/

void ma_fonction(void)                  /*La fonction*/
{
    printf("Hello world!\n");
}

int main(void)
{
    void (*pointeurSurFonction)(void);  /*Déclaration du pointeur*/

    pointeurSurFonction = ma_fonction;  /*Sauvegarde de l'adresse
de la fonction dans le pointeur adéquat*/

    return 0;
}
```

Fonction avec paramètres (avec arguments)

Une fonction est dite avec argument si dans la liste de ses arguments, il y en a, au moins un, différent du type `void`. Donc une fonction comme ceci :

Code : C

```
void ma_fonction(int argint);
```

Dans ce cas de figure, et par analogie au prototype de la fonction, la déclaration d'un pointeur sur ce type de fonctions est comme ceci :

Code : C

```
void (*pointeurSurFonction)(int)
```

En ce qui est des fonctions à plusieurs arguments, c'est le même principe. Donc pour une fonction comme ceci :

Code : C

```
void ma_fonction(int argint,char * argchar,size_t leng);
```

La déclaration de notre pointeur est :

Code : C

```
void (*pointeurSurFonction)(int,char *,size_t);
```

La sauvegarde de l'adresse de la fonction quant à elle, reste la même que dans le cas précédent  (ce sera toujours le cas d'ailleurs). A savoir :

Code : C

```
pointeurSurFonction = ma_fonction;
```

Fonctions avec retour

Une fonction est dite "avec retour" quand le type de retour est différent du `void` .

Fonctions sans arguments

La déclaration d'un pointeur sur une fonction du genre `int ma_fonction(void)` est de la forme :

Code : C

```
int (*pointeurSurFonction)(void);
```

Fonctions avec arguments

La déclaration d'un pointeur sur une fonction du genre `int * ma_fonction(double * f,int n)` est de la forme :

Code : C

```
int * (*pointeurSurFonction)(double*,int);
```

Appels de fonctions en utilisant leurs adresses

Notion de callback

J'ouvre une parenthèse pour parler un peu du concept callback :

C'est un concept qui consiste à attacher une fonction à un objet, je m'explique :
On peut imaginer que l'on a un objet du genre un bouton, et que l'on souhaite lui associer une fonction, qui sera appelée lorsqu'on appuie sur ce bouton. C'est ce qu'on appelle une fonction de rappel (callback).

Ceci ne peut être fait, que par utilisation des pointeurs de fonctions.

En champ d'applications classiques, on pourrait citer les threads, qui lors de la création, se voient attribuer une fonction qui sera appelée suite à leur activation.

Opérateur d'appel de fonctions

Comme vous avez pu le connaitre (ou pas), l'appel aux fonctions est caractérisé par la présence de parenthèses '()'. Dans le cas général, on appellera une fonction à partir de son pointeur ainsi :

(*nom_du_pointeur_sur_fonction)(**liste_d'arguments**)

La valeur de retour peut être récupérée ainsi :

variable_receptrice = (*nom_du_pointeur_sur_fonction)(**liste_d'arguments**)

Il est possible d'utiliser l'appel d'une fonction sous la forme "classique" à savoir **pointeurSurFonction(argument)** (sans les parenthèses et l'astérisque '*'), mais un tel appel n'explicite pas qu'il s'agit d'un pointeur sur fonction. Donc je ne le conseillerai pas dans le cas général.

Exemple 1 : Fonction sans retour et sans arguments

Soit la fonction :

Code : C

```c
void afficherBonjour(void)
{
     printf("Bonjour\n");
}
```

Afin d'appeler la fonction afficherBonjour à partir de son pointeur, il faut faire comme ceci :

Code : C

```c
int main (void)
{
     void (*pointeurSurFonction)(void);        /*déclaration du pointeur*/
     pointeurSurFonction = afficherBonjour;    /*Initialisation*/

     (*pointeurSurFonction)();                 /*Appel de la fonction*/

     return 0;
}
```

Résultat :

Code : Console

```
Bonjour
```

Exemple 2 : Fonction sans retour et avec arguments

Code : C

```c
void afficherBonjour(char * nom)
{
     printf("Bonjour %s\n",nom);
}

int main (void)
{
     void (*pointeurSurFonction)(char *);          /*déclaration du pointeur*/
     pointeurSurFonction = afficherBonjour;        /*Initialisation*/

     (*pointeurSurFonction)("zero");               /*Appel de la fonction*/

     return 0;
}
```

Résultat :

Code : Console

```
Bonjour zero
```

Exemple 3 : Fonction avec retour et sans arguments

Code : C

```c
int saisirNombre(void)
{
     int n;

     printf("Saisissez un nombre entier : ");
     scanf("%d",&n);

     return n;
}

int main (void)
{
     int (*pointeurSurFonction)(void);         /*déclaration du pointeur*/
     int nombre;

     pointeurSurFonction = saisirNombre;       /*Initialisation*/

     nombre = (*pointeurSurFonction)();        /*Appel de la fonction*/

     return 0;
}
```

Exemple 4 : Fonction avec retour et avec arguments

Code : C

```c
int saisirNombre(char * nom)
{
	int n;

	printf("Bonjour %s saisie un nombre entier : ",nom);
	scanf("%d",&n);

	return n;
}

int main (void)
{
	int (*pointeurSurFonction)(char *);		/*déclaration du pointeur*/
	int nombre;

	pointeurSurFonction = saisirNombre;		/*Initialisation*/

	nombre = (*pointeurSurFonction)("zero");	/*Appel de la fonction*/

	return 0;
}
```

Tableau de pointeurs sur fonctions

La déclaration d'un tableau de pointeurs sur fonction se fait généralement sous la forme suivante :

type_de_retour (* `nom_du_tableau_de_pointeurs_sur_fonctions` [**taille_du_tableau**]) (**liste_des_arguments**);

Comme vous pouvez le remarquer, c'est complètement identique à ce qu'a été présenté avant (sauf les [] pour préciser que c'est un tableau).

Son utilisation est soumise à toutes les règles d'utilisation des tableaux ordinaires que vous connaissez (ou êtes censé connaître), à savoir le premier indice est *0* et le dernier est *taille - 1* , tous les pointeurs qui seront stockés dedans doivent avoir le même type (les fonctions qu'on mettra dedans doivent avoir **le même prototype**, autrement dit, le même type de retour et les mêmes arguments).

Un exemple est en dernière partie du tutoriel.

Retourner un pointeur sur fonctions

Méthode normale

Si l'on souhaite retourner un pointeur sur fonctions dans une de nos fonctions, nous pouvons utiliser le pointeur générique présenté tout au début du tutoriel (rappelez-vous ) cependant une telle pratique relève du non-portable. c'est pourquoi nous allons utiliser le pointeur type sur fonctions ainsi :

type_de_retour_de_la_fonction_pointee (* nom_de_la_fonction_de_renvoi (**liste_arguments**))(**liste_arguments_fonction_pointee**)

Exemple pour une fonction sans arguments :

Code : C

```c
int fonction1(void)
{
	return 1;
}
```

```c
int (* fonction2(void))(void)
{
     return fonction1;         /*Ici le retour d'un pointeur sur
fonction*/
}
```

Exemple de fonction avec arguments :

Code : C

```c
int fonction1(double,double)
{
     return 1;
}

int (* fonction2(char str[]))(double,double)
{
     printf("%s",str);        /*Affichage de la chaine passée en
paramètre*/

     return fonction1;        /*Ici le retour d'un pointeur sur
fonction*/
}
```

Utilisation d'un typedef

Avec cette méthode, on va pouvoir déclarer un pointeur sur fonction tel que nous l'avons appris plus haut  , mais cette fois ci en mettant **typedef** à sa gauche, comme ceci :

typedef type_retour (* nom_du_pointeur) (**liste_arguments**)

Et ceci dans le champ des déclarations globales (c'est-à-dire en dehors des fonctions).

Exemple de fonction sans arguments :

Code : C

```c
typedef int (*ptrFonction)(void);

int fonction1(void)
{
     return 1;
}

ptrFonction fonction2(void)
{
     return fonction1;         /*Ici le retour d'un pointeur sur
fonction*/
}
```

Exemple de fonction avec arguments :

Code : C

```c
typedef int (*ptrFonction)(double,double);
```

```
int fonction1(double,double)
{
     return 1;
}

ptrFonction fonction2(char str[])
{
     printf("%s",str);       /*Affichage de la chaine passée en
paramètre*/

     return fonction1;       /*Ici le retour d'un pointeur sur
fonction*/
}
```

Passage de pointeur de fonctions en paramètre

Ici nous allons voir qu'il est possible de passer un pointeur sur fonction en paramètre à une autre fonction.

Deux méthodes existent, on peut citer en premier celle qu'on a utilisé jusqu'ici (la déclaration void (*ptr)(void)). La deuxième étant utiliser un **typedef** (d'ailleurs je vous la conseille vivement).

Méthode classique

Voici comment dire que c'est un pointeur sur fonction qu'on passe en paramètre :

Code : C

```
void fonction1(int n)
{
    printf("fonction 1 appel N° %d\n",n);
}

void fonction2(int n, void (*ptrfonction)(int))
{
     (*ptrfonction)(n);       /*Appel de la fonction pointée par
ptrfonction*/
}
```

On peut donc appeler notre fonction ainsi :

Code : C

```
fonction2(13,fonction1);
```

Ainsi le résultat sera :

Code : Console

```
fonction 1 appel N° 13
```

Utilisation de typedef

Bien que la méthode ci-dessus soit correcte, la deuxième que je vais vous présenter ici, va vous permettre d'utiliser le pointeur de

fonction comme n'importe quel autre type .

Et pour cela il nous suffirait de déclarer notre pointeur en typedef (comme vue dans la partie juste avant) :

Code : C

```c
typedef void (*ptrfonction)(int);
```

Et si je reprends le même exemple, ceci donnerait :

Code : C

```c
void fonction1(int n)
{
    printf("fonction 1 appel N° %d\n",n);
}

void fonction2(int n, ptrfonction ptr)
{
     (*ptr)(n);      /*Appel de la fonction pointée par ptrfonction*/
}

int main(void)
{
     fonction2(13,fonction1);
}
```

Utilité des pointeurs sur fonction

Dans le cas d'un programmeur débutant ou même intermédiaire, il est très rare qu'on se retrouve dans la nécessité d'utiliser ce genre de pointeurs.

Cependant, dans certaines architectures on peut être amené à faire le choix d'utiliser de tels appels de fonctions (à partir de leurs pointeurs), qui permettent d'avoir une flexibilité. contrairement aux autres appels statiques qui rigidifie notre architecture.

Dans la difficulté de trouver un exemple adéquat avec la difficulté de ce tutoriel, j'ai finalement choisi de vous présenter la version *callback* pour l'exercice de la calculatrice (vous vous en rappelez ?)

Pour ceux qui ne l'ont pas fait, j'en rappelle l'énoncé :

Citation : Exercice calculatrice

Il faut écrire un programme qui affiche un menu avec les calculs d'addition, soustraction, multiplication et division à l'utilisateur, et lui demande d'en choisir un. Ensuite il lui demande de saisir les opérandes.
Le programme devra faire le calcul choisi avec les opérandes entrées, et afficher le résultat.

Voici l'explication de la solution :

Les fonctions de calcul

Nous allons dans un premier temps écrire 4 fonctions traitants les différents calculs (addition, soustraction, multiplication et division) :

Code : C

```c
double addition(double n1,double n2)
```

```
{
    return n1 + n2;
}
double soustraction(double n1,double n2)
{
    return n1 - n2;
}
double multiplication(double n1,double n2)
{
    return n1 * n2;
}
double division(double n1,double n2)
{
    return n1 / n2;
}
```

Déclaration et initialisation d'un tableau de pointeurs sur fonctions

Maintenant nous allons créer un tableau de pointeurs sur fonctions, qu'on initialisera avec les fonctions de calcul qu'on vient d'écrire :

Code : C

```
double (*listeFonctions[4])(double,double) =
{addition,soustraction,multiplication,division};
```

Rappelez-vous, la déclaration d'un pointeur est `double (*listeFonctions)(double,double)`.
Donc la déclaration d'un tableau est de la forme `double (*listeFonctions[4])(double,double)`
Ce tableau est ensuite initialisé avec : `{addition,soustraction,multiplication,division};`

(Cet exemple nous a permis de voir comment utiliser un tableau de pointeurs de fonctions  chouette non ?).

Affichage du menu

Ensuite nous allons créer une fonction qui va afficher un menu, proposant à l'utilisateur de choisir une opération :

Code : C

```
double (* affichMenu(void))(double,double)        /*Le retour
correspondant à un pointeur sur fonction*/
{
    int choix;                                    /*La variable pour
le choix*/
    do{
         printf("--------------------------------MENU----------------
----------------------\n");
         printf("Veuillez choisir une operation (en choisissant un
nombre entre 1 et 4) :\n");
         printf("1 pour addition\n");
         printf("2 pour soustraction\n");
         printf("3 pour multiplication\n");
         printf("4 pour division\n");
         printf("Votre choix : ");
         scanf("%d",&choix);

    }while(choix < 1 || choix > 4 );              /*En cas d'erreur de
saisie*/
```

```
    return listeFonctions[choix - 1];           /*On renvoi le
pointeur sur la fonction de calcul choisie*/
}
```

Cette même fonction contrôle la saisie, si elle est correcte, elle envoie un pointeur sur la fonction choisie.

La fonction main

Maintenant il ne nous reste plus que remplir la fonction main :

Code : C

```
int main (void)
{
    double (*fonctionDeCalcul)(double,double);        /*déclaration
du pointeur*/
    double n1,n2;

    fonctionDeCalcul = affichMenu();                  /*On charge la
fonction de calcul choisie*/

    printf("Saisissez les operandes : ");             /*On lit les
opérandes de notre calcul*/
    scanf("%lf",&n1);
    scanf("%lf",&n2);

    printf("le resultat du calcul est :
%f\n",(*fonctionDeCalcul)(n1,n2));  /*On appelle la fonction
choisie et on affiche le résultat.*/
    return 0;
}
```

Le programme complet :

Secret (cliquez pour afficher)

Code : C

```
#include <stdio.h>
//---------Les prototypes-----------
double addition(double n1,double n2);
double soustraction(double n1,double n2);
double multiplication(double n1,double n2);
double division(double n1,double n2);
double (* affichMenu(void))(double,double);

//------Les fonctions de calcul-------
double addition(double n1,double n2)
{
    return n1 + n2;
}
double soustraction(double n1,double n2)
{
    return n1 - n2;
}
double multiplication(double n1,double n2)
{
    return n1 * n2;
```

```
}
double division(double n1,double n2)
{
    return n1 / n2;
}

//------Le tableau de pointeur sur fonctions (global)------
double (*listeFonctions[4])(double,double) =
{addition,soustraction,multiplication,division};

//--------Fonction du menu--------
double (* affichMenu(void))(double,double)       /*Le retour
correspondant à un pointeur sur fonction*/
{
    int choix;                                   /*La variable
pour le choix*/
    do{
         printf("--------------------------------MENU--------------
------------------------\n");
         printf("Veuillez choisir une operation (en choisissant un
nombre entre 1 et 4) :\n");
         printf("1 pour addition\n");
         printf("2 pour soustraction\n");
         printf("3 pour multiplication\n");
         printf("4 pour division\n");
         printf("Votre choix : ");
         scanf("%d",&choix);

    }while(choix < 1 || choix > 4 );        /*En cas d'erreur de
saisie*/

    return listeFonctions[choix - 1];       /*On renvoi le
pointeur sur la fonction de calcul choisie*/
}

//--------Le main----------
int main(void)
{
     double (*fonctionDeCalcul)(double,double);
/*déclaration du pointeur*/
     double n1,n2;                                      /*Les
opérandes*/

     fonctionDeCalcul = affichMenu();                   /*On lit la
fonction choisie*/

     printf("Saisissez les operandes : ");              /*On
demande la saisie des opérandes*/
     scanf("%lf",&n1);
     scanf("%lf",&n2);

     printf("le resultat du calcul est :
%f\n",(*fonctionDeCalcul)(n1,n2));   /*On affiche le résultat*/

     return 0;
}
```

Pour résumer, nous allons retenir les choses suivantes :

- Le pointeur void ne peut pas contenir l'adresse d'une fonction de manière portable et sans pertes.
- Le nom d'une fonction permet d'avoir un pointeur sur cette fonction.
- La déclaration d'un pointeur de fonction doit respecter le prototype de la fonction sur laquelle il va pointer.

Vous savez à présent à quoi correspond un pointeur sur fonction, alors faites-en bon usage  .