Les instructions du langage C

Les expressions et les opérateurs

Définition de fonctions

Accès rapide :

Les blocs d'instructions
Les instructions conditionnelles
L'instruction *if*
L'instruction *switch*
Les instructions de boucles
L'instruction *for*
L'instruction *while*
L'instruction *do while*
Les instructions de débranchement
L'instruction *break*
L'instruction *continue*
L'instruction *goto*

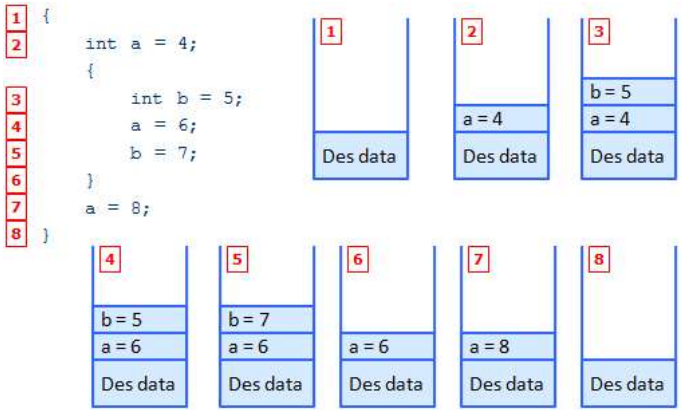
Les blocs d'instructions

Comme son nom le laisse présager, le bloc d'instructions peut contenir plusieurs instructions (il peut aussi en contenir qu'une unique). Le bloc commence par une accolade ouvrante et se termine par une accolade fermante. L'exemple ci-dessous montre un bloc d'instructions vous demandant de saisir votre nom : ce bloc est constitué de quatre lignes de code.

```
{
    printf( "Veuillez renseigner votre nom
    char buffer[80];
    scanf( "%s", buffer );
    printf( "Hello %s\n" );
}
```

Il faut savoir qu'un bloc d'instructions impose une durée de vie aux variables qui sont déclarées à l'intérieur de ce bloc. En fait une telle variable (dite locale) est stockée dans un espace de mémoire particulier appelé la pile d'exécution (ou stack, en anglais). L'état de la pile évolue au cours du temps en fonction des variables de l'on défini.

Un autre point est aussi important à savoir : la taille de la pile d'exécution est limitée. Si vous saturez cette pile d'exécution, une erreur FATALE sera déclenchée. En réalité, un programme complexe n'a pas qu'une unique pile d'exécution. Effectivement, cette notion de pile est en fait liée à la notion de threads (aussi appelé fil d'exécution) : il existe donc une pile d'exécution par thread.



Evolution de la pile d'exécution au cours du temps

Le diagramme ci-dessus vous montre comment les variables naissent, évoluent et meurent au cours de l'exécution séquentielle de vos instructions. Au final de l'exécution des lignes de code proposées à titre d'exemple, l'état de la pile sera donc revenue dans son état initiale, et les variables locales manipulées seront donc définitivement perdues. Notez bien qu'à partir de la ligne 6, la variable b est perdue et ne sera donc plus accessible. Si vous tentez néanmoins de la manipuler, le compilateur vous renverra une erreur de compilation.

Pour les instructions, qui vont suivrent, il est important de noter qu'elles acceptent normalement qu'une seule sous instruction à exécuter. Sauf qu'un bloc d'instruction est vu comme étant "une" instruction. On peut donc utiliser le bloc pour demander à une instruction de faire plusieurs choses. Par exemple, dans l'exemple ci-dessous, la seconde instruction conditionnelle `if` n'exécutera que le premier affichage si la condition est vrai. Le second affichage sera systématiquement exécuté que la condition soit vrai ou non. Notez que pour le premier `if`, un bloc a bien été utilisé, sans quoi le `return` aurait été systématique.

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


int main( int argc, char *argv[] ) {

    if ( argc != 2 ) {
        printf( "Usage: Essai numValue\n"
        return EXIT_SUCCESS;
    }

    int a = atoi( argv[1] );
    if ( a < 5 )
        printf( "First display\n" );
        printf( "Second display\n" );


    return EXIT_SUCCESS;
}
```

Testons ce petit programme

```
$> gcc -Wall -o Sample Sample.c
$> ./Sample 3
First display
Second display
$> ./Sample 6
Second display
$>
```

Si ce que vous vouliez était de réaliser les deux affichages uniquement quand la condition est vrai, il aurait plutôt fallu écrire le code suivant.

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


int main( int argc, char *argv[] ) {
```

```
    if ( argc != 2 ) {
        printf( "Usage: Essai numValue\n"
        return EXIT_SUCCESS;
    }

    int a = atoi( argv[1] );
    if ( a < 5 ) {
        printf( "First display\n" );
        printf( "Second display\n" );
    }

    return EXIT_SUCCESS;
}
```

Testons cette variante de notre programme.

```
$> gcc -Wall -o Sample Sample.c
$> ./Sample 3
First display
Second display
$> ./Sample 6
$>
```

Cette règle (le bloc d'instructions compte pour une instruction) sera utilisable pour toutes les instructions qui acceptent une chose à réaliser : `if`, `for`, `while`, `do while`. Seule l'instruction `switch` dérogera un peu à la règle. Personnellement, je vous conseille de systématiquement utiliser un bloc d'instructions. Cela vous évitera de nombreuses erreurs.

Les instructions conditionnelles

Il existe deux instructions conditionnelles (permettant d'exécuter différentes actions en fonction de certaines conditions) : l'instruction `if` et l'instruction `switch`. Regardons de plus près chacune d'entre elle.

L'instruction if

```
if ( condition ) statement

if ( condition ) { statement, ... }

if ( condition ) statement else statement

if ( condition ) { statement, ... } else { statement, ... }
```

L'instruction `if` permet d'exécuter un code si une condition particulière est constatée. Il est aussi possible de spécifier un code à exécuter si la condition n'est pas vraie : dans ce cas, le code associé est introduit par le mot clé `else`.

Que la condition soit vraie, ou non, vous pouvez spécifier une instruction à exécuter ou bien un bloc d'instructions. Dans ce dernier cas, le bloc est introduit par une accolade ouvrante et se termine par une accolade fermante.

Il est à noter que C ne définit pas le mot clé `then` : il faut donc que le compilateur puisse détecter ou se termine la condition à évaluer. C'est pour cela, qu'en C, vous vous devez obligatoirement de parenthéser la condition.

Voici un exemple de code.

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

int main( int argc, char * argv[] ) {
    /* argc == argument counter */
    /* argv == argument values */

    if ( argc == 1 ) {
        printf( "Usage: commandName folder
        exit( 0 );
    } else {
        printf( "You have provided one par
        /* Notice that argv[0] is the comm
```

```
    }

    return 0;
}
```

L'instruction *switch*

```
switch ( expression ) {
    case value1:
        statement;
        [statement]...
        break;
    case value2:
        statement;
        [statement]...
        break;
    default:
        statement;
        [statement]...
}
```

L'instruction switch permet, comme l'instruction if, de déclencher des traitements en fonction d'une condition (d'un test). D'un certain point de vue, cette instruction est similaire à plusieurs if imbriqués. Néanmoins les performances sont souvent meilleures avec un switch car un tableau de pointeurs, contenant les adresses des codes à exécuter, est calculé une fois pour toute. Ainsi, quelque soit la valeur considérée, le temps nécessaire à trouver le code à exécuter est constant, contrairement à plusieurs instructions if imbriquées.

La sélection du bloc d'instruction à exécuter s'effectue grâce à la valeur d'une expression. Cette expression doit calculer une valeur dont le type fait partie de la liste suivante : char, short, int, long, un pointeur ou un type énumérés.

Notez que le bloc d'instruction à exécuter ne nécessite pas d'accolades. Néanmoins, le fait d'utiliser des accolades permet de mieux controler la durée de vie des variables qui pourraient être définies dans un switch.

L'utilisation de l'instruction switch est souvent couplée à l'instruction break. Effectivement, si un bloc d'instructions est exécuté et si celui-ci ne se temine pas par l'instruction break, alors l'exécution du bloc suivant (dans l'ordre de déclaration) sera lancé et ainsi de suite. Cela est pratique pour définir un même traitement pour plusieurs cas.

Voici un exemple de code.

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

int main( int argc, char * argv[] ) {

    if ( argc == 1 ) {
        printf( "Usage: commandName intege
        exit( 0 );
    }
    int value = atoi( argv[1] );

    switch( value ) {
        case 1:
            printf( "One\n" );
            break;
        case 2:
            printf( "Two\n" );
            break;
        case 3:
            printf( "Three\n" );
            break;
        default:
            printf( "Other value\n" );
    }

    return 0;
}
```

Les instructions de boucles

Le langage C met trois instructions de boucles à votre disposition : l'instruction `for`, l'instruction `while` et l'instruction `do while`. Etudions les une à une.

L'instruction *for*

```
for ( init; condition; increment ) statement;

for ( init; condition; increment ) {
    [statement]...
}
```

L'instruction `for` permet d'introduire une boucle dans votre programme. Cette instruction nécessite que vous spécifiez (entre les parenthèses) trois expressions (séparées par des caractères ;). Ces trois expressions vous permettront de contrôler l'exécution de votre boucle.

La première expression permet d'initialiser la valeur initiale de votre compteur de boucle.

La second expression permet de savoir si un nouveau tour de boucle doit être réalisé ou non (On parle de condition de rebouclage) : tant que la valeur calculée par cette expression est vrai (!= 0), un nouveau tour de boucle sera réalisé. Si la valeur calculée est fausse (== 0), la boucle s'arretera. Cette expression est évaluée y compris avant le premier tour de boucle. Si dès la première évaluation, la valeur calculée est fausse, aucun tour de boucle ne sera réalisé.

Accueil Langage C

Notre page Facebook sur C
Notre groupe Facebook sur C

Le tutoriel sur le langage C
Introduction et historique
Compilation d'un programme C
Présentation de l'atelier Eclipse/CDT
Le préprocesseur C
Les types et les variables du langage C
Les expressions du langage C
Les instructions du langage C
Définition de fonctions en C
Utilisation de pointeurs
Gestion dynamique de la mémoire
Manipulation de listes chaînées
Les pointeurs sur fonctions
Quelques mots clés complémentaires
Les nouveautés introduites par C99
Les nouveautés introduites par C11
Les principales options de gcc

Les instructions du préprocesseur

Les instructions C

La librairie <assert.h>

La librairie <complex.h>

La librairie <ctype.h>

La librairie <errno.h>

La librairie <fenv.h>

La librairie <math.h>

La librairie <signal.h>

La librairie <stdarg.h>

La librairie <stdbool.h>

Voici un exemple de code.

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

int main( int argc, char * argv[] ) {

    int counter;
    for( counter=0; counter<argc; counter+
        printf( "Parameter %d == %s\n", co
    }

    return 0;
}
```

L'instruction *while*

```
while ( condition ) statement;

while ( condition ) {
    [statement]...
}
```

A l'instar de l'instruction `for`, le `while` permet aussi d'introduire une boucle dans votre programme. La boucle sera exécutée tant que la condition (forcément placée entre parenthèses) sera vraie (donc différente de 0).

Comme les autres instructions du langage C, soit vous avez une seule instruction à exécuter à chaque tour de boucle et dans ce cas les accolades sont facultatives, soit vous y placez plusieurs instructions et là elles deviennent obligatoires.

Voici un exemple de code.


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

int main( int argc, char * argv[] ) {

    int counter = 0;
    while( counter < argc ) {
```

La librairie <stdio.h>

La librairie <stdlib.h>

La librairie <string.h>

La librairie <threads.h>

La librairie <time.h>

Quelques librairies non standards

Testez vos connaissances en C


```
        printf( "Parameter %d == %s\n", co
        counter++;
    }

    return 0;
}
```

L'instruction *do while*

```
do
    statement;
while ( condition );

do {
    [statement]...
} while ( condition );
```

L'instruction `do / while` est très simimaire à l'instruction `while`. Néanmoins, la condition est exécutée en fin de boucle. L'instruction `do / while` garantira donc qu'au moins un tour de boucle soit exécuté, contrairement à l'instruction `while` qui peut ne pas faire le moindre tour de boucle (si la condition est immédiatement vraie).

Voici un exemple de code.

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

int main() {

    char buffer[80];
    do {
```


Accueil Langage C

Notre page Facebook sur C

Notre groupe Facebook sur C

Le tutoriel sur le langage C

Introduction et historique
Compilation d'un programme C
Présentation de l'atelier Eclipse/CDT
Le préprocesseur C
Les types et les variables du langage C
Les expressions du langage C
Les instructions du langage C
Définition de fonctions en C
Utilisation de pointeurs
Gestion dynamique de la mémoire
Manipulation de listes chaînées
Les pointeurs sur fonctions
Quelques mots clés complémentaires
Les nouveautés introduites par C99
Les nouveautés introduites par C11
Les principales options de gcc

Les instructions du préprocesseur

Les instructions C

La librairie <assert.h>

La librairie <complex.h>

La librairie <ctype.h>

La librairie <errno.h>

La librairie <fenv.h>

La librairie <math.h>

La librairie <signal.h>

La librairie <stdarg.h>

La librairie <stdbool.h>

```
    return 0;
}
```

Les instructions de débranchement

L'instruction *break*

L'instruction `break` est utilisée pour terminer une boucle. Très souvent son utilisation est précédée d'une instruction conditionnelle `if`.

Notez aussi que cette instruction peut aussi être utilisée pour terminer l'exécution d'une branche conditionnelle pour l'instruction `switch` : je vous renvoie sur la documentation de cette instruction pour de plus amples informations à ce sujet.

Voici un exemple de code. Il recherche si l'option `-h` est présente sur la liste des arguments passéssur la ligne de commande lors du démarrage du programme.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main( int argc, char * argv[] ) {

    int counter;
    int helpRequested = 0;

    argc--;   argv++;

    if ( argc == 0 ) {
        printf( "Usage: commandName argume
        exit( 0 );
    }

    for( counter=0; counter<argc; counter+
        if ( strcmp( argv[counter], "-h")
            helpRequested = 1;
            break; /* No other loop requir
```

La librairie <stdio.h>

La librairie <stdlib.h>

La librairie <string.h>

La librairie <threads.h>

La librairie <time.h>

Quelques librairies non standards

Testez vos connaissances en C


```
        }
    }

    if ( helpRequested ) {
        printf( "Help requested\n" );
    }

    /* . . . */

    return 0;
}
```

L'instruction *continue*

L'instruction `continue` permet d'intérrompre le tour de boucle en cours est de passer au suivant. Dans le cas de l'utilisation de l'instruction `continue` dans une bouble `for`, l'incrémentation de la variable de boucle sera bien effectuée.

Voici un exemple de code. Il permet d'ignorer toutes les options (dont le premier caractère est un -) passées en tant qu'argument sur la ligne de commande de démarrage du programme.

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

int main( int argc, char * argv[] ) {

    int counter = 0;

    argc--;   argv++;

    for( counter=0; counter<argc; counter+
        if ( argv[counter][0] == '-') {
            printf( "Skip option %s\n", ar
            continue;
        }
        printf( "Handle argument %s\n", ar
    }

    // . . .

    return 0;
}
```


Accueil Langage C

Notre page Facebook sur C

Notre groupe Facebook sur C

Le tutoriel sur le langage C
Introduction et historique
Compilation d'un programme C
Présentation de l'atelier Eclipse/CDT
Le préprocesseur C
Les types et les variables du langage C
Les expressions du langage C
Les instructions du langage C
Définition de fonctions en C
Utilisation de pointeurs
Gestion dynamique de la mémoire
Manipulation de listes chaînées
Les pointeurs sur fonctions
Quelques mots clés complémentaires
Les nouveautés introduites par C99
Les nouveautés introduites par C11
Les principales options de gcc

Les instructions du préprocesseur

Les instructions C

La librairie <assert.h>

La librairie <complex.h>

La librairie <ctype.h>

La librairie <errno.h>

La librairie <fenv.h>

La librairie <math.h>

La librairie <signal.h>

La librairie <stdarg.h>

La librairie <stdbool.h>

L'instruction *goto*

L'instruction `goto` permet de réaliser ce qu'on appelle un débranchement. Un débranchement permet de se repositionner sur une autre section de code a exécuter, introduite par une étiquette, au lieu de poursuivre une exécution séquentielle.

Une étiquette se définie avec un nom suivi d'un caractère :. Normalement, une étiquette est placée en colonne 0 du fichier de texte.

En fait quasiment toutes les instructions C effectuent de débranchement. Pour les autres instructions, les choses sont contrôlée par le compilateur, mais ce qui n'est pas le cas avec `goto`. Effectivement, si l'étiquette de destination n'est pas placée dans le même bloc que l'instruction `goto`, alors vous pouvez désynchroniser la pile d'exécution et donc aboutir à un crash de l'application. **C'est pour cette raison qu'il est très fortement conseillé de ne pas utiliser l'instruction `goto` au profit des autres instructions.**

Voici un petit exemple de code.


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

int main( int argc, char * argv[] ) {
```

La librairie <stdio.h>

La librairie <stdlib.h>

La librairie <string.h>

La librairie <threads.h>

La librairie <time.h>

Quelques librairies non standards

Testez vos connaissances en C


```
    int counter = 0;

    argc--;   argv++;

    if ( argc == 0 ) {
        printf( "Usage: commandName argume
        exit( 0 );
    }

loopBegin:
    if ( counter == argc )
        goto loopEnd;

    printf( "Handle argument %s\n", argv[c
    counter++;
    goto loopBegin;
loopEnd:

    // . . .

    return 0;
}
```

Les expressions et les opérateurs

Définition de fonctions

Le tutoriel sur le langage C
- Introduction et historique
- Compilation d'un programme C
- Présentation de l'atelier Eclipse/CDT
- Le préprocesseur C
- Les types et les variables du langage C
- Les expressions du langage C
- Les instructions du langage C
- Définition de fonctions en C
- Utilisation de pointeurs
- Gestion dynamique de la mémoire
- Manipulation de listes chaînées
- Les pointeurs sur fonctions
- Quelques mots clés complémentaires
- Les nouveautés introduites par C99
- Les nouveautés introduites par C11
- Les principales options de gcc

Les instructions du préprocesseur

Les instructions C

La librairie <assert.h>

La librairie <complex.h>

La librairie <ctype.h>

La librairie <errno.h>

La librairie <fenv.h>

La librairie <math.h>

La librairie <signal.h>

La librairie <stdarg.h>

La librairie <stdbool.h>