

## Pratique du C

### Classes d'allocation, contexte et passage de paramètres par la pile

Licence Informatique — Université Lille 1  
Pour toutes remarques : Alexandre.Sedoglavic@univ-lille1.fr

Semestre 5 — 2010-2011

V0 (19-10-2010)

Les variables externes (extern) sont :

- ▶ allouées en zone statique de données (dans un segment de données) ;
- ▶ allouées à la compilation (valeur par défaut 0) ;
- ▶ durée de vie du programme ;
- ▶ visibles depuis toutes les fonctions.

Les variables statiques (static) sont :

- ▶ allouées comme les variables externes ;
- ▶ et si elles sont définies :
  - ▶ à l'extérieur de toute fonction, elles sont visibles depuis les fonctions déclarées dans le fichier source les contenant ;
  - ▶ à l'intérieur d'une fonction, elles sont visibles depuis la fonction seulement, mais reste allouées en dehors de l'exécution de la fonction (valeur conservée entre les différents appels).

V48 (19-10-2010)

Par exemple, on peut avoir le code suivant :

```
int global = 1; /* d'efinition d'une variable
               externe (globale) */

extern int extern_global ; /* d'eclaration d'une variable
                           globale d'un autre fichier (externe) */

static int global_privée = 2 ; /* globale au fichier,
                               invisible depuis d'autres fichiers (statique) */

int
fonction(int param) { /* param\etre (automatique) */
    auto int local = 3 ; /* variable automatique (locale) */
    /* le mot clef auto ne sert \a rien */
    static int local_statique = 4 ; /* variable
                                     statique (locale) valeur inchang\ee entre 2 appels */

    register int i = 5 ; /* variable register (locale) */

    return i++ ;
}
```

V48 (19-10-2010)

## Les classes d'allocation des variables

En C, les variables ont pour attribut :

- ▶ leur nom : un identificateur ;
- ▶ leur type : type de base ou défini par l'utilisateur ;
- ▶ une classe d'allocation indiquant :
  - ▶ le type de l'emplacement mémoire où est allouée la variable ;
  - ▶ sa durée de vie ;
  - ▶ sa visibilité par les différentes fonctions.

Il y a 5 classes d'allocation :

externe    automatique    statique    "register"    volatile

Les variables automatiques (auto) sont :

- ▶ allouées dynamiquement sur la pile (valeur initiale indéterminée) ;
- ▶ allouées à chaque entrée dans la fonction ou le bloc où la variable est définie (paramètres, variables locales) ;
- ▶ durée de vie de la fonction ou du bloc ;
- ▶ visibles uniquement depuis la fonction ou le bloc.

Les variables de registres (register) sont :

- ▶ allouées si possible dans un registre du processeur ;
- ▶ des variables de type simple uniquement ;
- ▶ des variables de classe automatique uniquement ;
- ▶ et ne possèdent pas d'adresse.

Le code assembleur correspondant est :

```
int global = 1;                                .data
                                               .globl global
extern int extern_global ;                    global:
                                               .long    1
static int global_privée = 2 ;                global_privée:
                                               .long    2
int fonction(int param) {                     local_statique.0:
    int local = 3 ;                          .long    4
                                               .text
    static int local_statique = 4 ;          .globl fonction
    fonction:
        register int i = 5 ;                 pushl   %ebp
                                               movl   %esp, %ebp
        return i++ ;                         subl   $4, %esp
    }                                         movl   $3, -4(%ebp)
                                               movl   $5, %eax
                                               leave
                                               ret
```

V48 (19-10-2010)

<div>Pratique du C</div> <div>Classes d'allocation, contexte et passage de paramètres par la pile</div>	<pre>extern int ailleurs(int) ;  static int foo(void){     return 1 ; }  int bar(void){     return 1 ; }</pre>	<pre>.text .type    foo,@function  pushl    %ebp movl     %esp, %ebp movl     \$1, %eax popl     %ebp ret  .Lfe1: .size    foo, .Lfe1-foo .globl   bar .type    bar,@function  bar: pushl    %ebp movl     %esp, %ebp movl     \$1, %eax popl     %ebp ret .Lfe2: .size    bar, .Lfe2-bar</pre>
Les classes d'allocation des variables		
Effet des mots clef static et extern sur les fonctions		
Ordre d'évaluation des paramètres		
Fonction à nombre variable de paramètres		
Définition : un contexte dans la pile d'exécution (stack frame)		
setjmp/longjmp		
Passage de paramètres		

V48 (19-10-2010)

<div>Pratique du C</div> <div>Classes d'allocation, contexte et passage de paramètres par la pile</div>	<p>L'usage de la classe d'allocation volatile supprime l'optimisation concernant la variable ainsi qualifiée. Le code :</p> <pre>static volatile int foo;  void bar (void) {     foo=0 ;     while(foo!=1)         continue ;     return ; }</pre>	
Les classes d'allocation des variables		
Effet des mots clef static et extern sur les fonctions		
Ordre d'évaluation des paramètres		
Fonction à nombre variable de paramètres		
Définition : un contexte dans la pile d'exécution (stack frame)		
setjmp/longjmp		
Passage de paramètres		

correspond donc au cas de figure au la variable foo est partagée (par exemple entre deux processus légers ou dans un segment de mémoire partagé — cf. cours Pratique des Systèmes).

Sans volatile, le compilateur simplifie ce code en boucle infinie (comme foo n'est pas modifié, cette variable est supprimée).

V48 (19-10-2010)

<div>Pratique du C</div> <div>Classes d'allocation, contexte et passage de paramètres par la pile</div>	<p>Attention aux surprises lors de l'évaluation des paramètres à transmettre :</p> <pre>#include &lt;stdio.h&gt;  int main(void){     int foo = 'a' ;      printf("le premier argument %c et le second %c\n",foo,foo++);     return 0 ; }  \$ a.out le premier argument b et le second a</pre>	<pre>.rodata .LC0: .string "le premier argument %c..." .text .globl main main: pushl    %ebp movl     %esp, %ebp subl     \$8, %esp movl     \$97, -4(%ebp) subl     \$4, %esp movl     -4(%ebp), %eax pushl    %eax leal     -4(%ebp), %eax incl     (%eax) pushl    -4(%ebp) pushl    \$.LC0 call     printf addl     \$16, %esp movl     \$0, %eax leave    ret</pre>
Les classes d'allocation des variables		
Effet des mots clef static et extern sur les fonctions		
Ordre d'évaluation des paramètres		
Fonction à nombre variable de paramètres		
Définition : un contexte dans la pile d'exécution (stack frame)		
setjmp/longjmp		
Passage de paramètres		

V48 (19-10-2010)

<div>Pratique du C</div> <div>Classes d'allocation, contexte et passage de paramètres par la pile</div>		
Les classes d'allocation des variables		
Effet des mots clef static et extern sur les fonctions		
Ordre d'évaluation des paramètres		
Fonction à nombre variable de paramètres		
Définition : un contexte dans la pile d'exécution (stack frame)		
setjmp/longjmp		
Passage de paramètres		

Les variables volatiles sont susceptibles d'être modifiées indépendamment du code les déclarant. Considérons l'exemple suivant (seul dans un fichier sources) :

```
static int foo; /* foo pourrait \etre un pointeur sur
un segment de m\emoire partag\ee */

void
bar(void){
    foo=0 ;
    while(foo!=1)
        continue ;
}
```

Un compilateur en optimisant ce code remplacera la boucle par while(1) car la classe d'allocation static lui assure que seule la fonction bar peut modifier foo. Cependant, l'emplacement mémoire associé à la variable pourrait être partagé — et donc modifiable — par un autre processus (voir l'unité d'enseignement *Pratique des systèmes*) et l'optimisation du compilateur ne pas correspondre à la volonté du programmeur.

Une variable dont le type est qualifié par const ne peut pas être modifiée.

Ce qualificatif permet au programmeur de s'assurer de ne pas modifier des variables passées par référence comme par dans le cas suivant :

```
#include <string.h>
int strcmp(const char *s1, const char *s2);
```

Ainsi, on est sur que cette fonction ne va pas modifier les chaînes de caractères passées en arguments.

Remarquons que les qualificateurs de types peuvent être utilisés finement :

```
const char c ; /* caract\ere constant */
const char *s ; /* pointeur vers caract\eres constants */
char const *s ; /* pointeur constant vers caract\eres */
const char * const s ; /* pointeur constant
vers caract\eres constants */
```

Il est possible de déclarer une fonction comme ayant un nombre variable de paramètres en « déclarant » les paramètres optionnels par l'unité lexicale ... (3 points à la suite) :

```
int
foo
(char *par_obl, ...)
{
    return 0 ;
}
```

Une fonction peut avoir à la fois des paramètres obligatoires et des paramètres optionnels, les paramètres obligatoires apparaissant en premier et l'unité lexicale ... apparaissant en dernière position dans la liste de déclaration des paramètres formels.

Généralement, un paramètre obligatoire indique le nombre et le type des paramètres optionnels comme dans le cas de printf :

```
printf("premier argument %c et second %c\n",foo,foo++) ;
```

www.fil.univ-lille1.fr/~sedoglav/C/Cours09.pdfV48 (19-10-2010)

Pratique du C

Classes d'allocation, contexte et passage de paramètres par la pile

Les classes d'allocation des variables

Effet des mots clef static et extern sur les fonctions

Ordre d'évaluation des paramètres

Fonction à nombre variable de paramètres

Définition : un contexte dans la pile d'exécution (stack frame)

setjmp/longjmp

Passage de paramètres

```
int somme(int nbpar, ...){
    int *pt = &nbpar ; /* on fait pointer pt
                        sur le premier param\etre */

    int res = 0;

    for(;nbpar>0;nbpar--){
        pt++      ; /* on passe au param\etre suivant */
        res += *pt ;
    }
    return res ;
}

int
main(void){
    return somme(3,1,2,3)+somme(4,4,5,3,1) ;
}
```

www.fil.univ-lille1.fr/~sedoglav/C/Cours09.pdfV48 (19-10-2010)

Pratique du C

Classes d'allocation, contexte et passage de paramètres par la pile

Les classes d'allocation des variables

Effet des mots clef static et extern sur les fonctions

Ordre d'évaluation des paramètres

Fonction à nombre variable de paramètres

Définition : un contexte dans la pile d'exécution (stack frame)

setjmp/longjmp

Passage de paramètres

## Un exemple d'utilisation (paramètres optionnels de même type)

```
int somme(int nbpar, ...){
    int *pt = &nbpar ; /* on fait pointer pt
                        sur le premier param\etre */

    int res = 0;

    for(;nbpar>0;nbpar--){
        pt++      ; /* on passe au param\etre suivant */
        res += *pt ;
    }
    return res ;
}

int
main(void){
    return somme(3,1,2,3)+somme(4,4,5,3,1) ;
}
```

On peut étendre la notion de contexte en y associant en plus des informations concernant la pile d'exécution, l'état des registres du processeur (%EAX, %EIP, %CS, etc).

Ce faisant, on peut faire des branchements non-locaux (i.e. des branchements à un endroit presque arbitraire du code) sans utiliser goto. Deux fonctions de la librairie standard sont dédiées à cet effet. Schématiquement,

- ▶ setjmp mémorise son contexte juste avant son RET ;
- ▶ longjmp permet de rétablir le contexte mémorisé en plaçant son second argument dans %EAX.

```
#include <setjmp.h>
#include <stdio.h>
int main (void){
    jmp_buf env;
    int i = setjmp(env) ; /* au premier appel setjmp retourne 0 */
    printf("i = %d\n",i);
    if(i == 2) return 0 ;
    longjmp(env,2) ; /* on branche sur setjmp qui retourne 2 */
    return 1 ; /* cette instruction n'est jamais ex\ecut\ee */
}
```

## Passage de paramètres par copie : une copie est faite sur la pile

```
.globl main
main:
    pushl %ebp
    movl  %esp, %ebp
    subl  $8, %esp
    andl  $-16, %esp
    movl  $1, -4(%ebp)
    movl  $1, -8(%ebp)
    subl  $8, %esp

    int main(void){
        int a = 1 ;
        int b = 1 ;

        return 0 ;
    }

    movl  $0, %eax
    leave
    ret
```

On appelle contexte d'un appel de fonction dans la pile d'exécution la partie de la pile associée :

- ▶ paramètres d'appels ;
- ▶ adresse de retour et ancien pointeur de contexte %EBP ;
- ▶ variables automatiques de la fonction.

Les différentes portions de pile correspondant aux différents contextes d'exécution peuvent être obtenues dans gdb par :

- ▶ backtrace : les contextes disponibles  
(gdb) backtrace  
#0 traduction (code=0xbffff660 "oeu") at SMS.c:12  
#1 0x08048532 in main () at SMS.c:45  
#2 0x400327f7 in \_\_libc\_start\_main
- ▶ info frame nb : qui affiche un contexte  
(gdb) info frame 2  
Stack frame at 0xbffff6f8: eip = 0x400327f7 in \_main;  
saved eip 0x8048301 caller of frame at 0xbffff6d8  
Arglist at 0xbffff6f8, args:  
Locals at 0xbffff6f8, Previous frame's sp in esp  
Saved registers: ebp at 0xbffff6f8, eip at 0xbffff6fc

Dans l'exemple précédent, setjmp et longjmp sont utilisées dans la même fonction mais généralement, ces fonctions sont utilisées pour la gestion d'erreurs et la programmation des systèmes (signal).

**Limitation** : longjmp ne permet pas de revenir à n'importe quel point mémorisé par l'appel setjmp ; ce n'est possible que si la fonction qui a exécutée le setjmp(env) n'est pas terminée car l'état de la pile n'est pas mémorisé (seuls les registres le sont). Sur une architecture de type intel, on a :

```
# if __WORDSIZE == 64
typedef long int __jmp_buf[8];
# else
typedef int __jmp_buf[6];
# endif
struct __jmp_buf_tag{
    __jmp_buf __jmpbuf; /* pour stocker les registres */
    int __mask_was_saved; /* pour d'autres usages syst\eme */
    __sigset_t __saved_mask; /* li\es aux signaux (cf. PDS) */
};
typedef struct __jmp_buf_tag jmp_buf[1];
extern int setjmp (jmp_buf __env) ;
```

```
.text
.globl main
main:
    pushl %ebp
    movl  %esp, %ebp
    subl  $8, %esp
    andl  $-16, %esp
    movl  $1, -4(%ebp)
    movl  $1, -8(%ebp)
    subl  $8, %esp
    pushl -8(%ebp)
    pushl -4(%ebp)
    call  PER
    addl  $16, %esp
    movl  $0, %eax
    leave
    ret

int main(void){
    int a = 1 ;
    int b = 1 ;
    PER(a,b) ;
    return 0 ;
}
```

www.fil.univ-lille1.fr/~sedoglav/C/Cours09.pdfV48 (19-10-2010)

Pratique du C

Classes d'allocation, contexte et passage de paramètres par la pile

Les classes d'allocation des variables

Effet des mots clef static et extern sur les fonctions

Ordre d'évaluation des paramètres

Fonction à nombre variable de paramètres

Définition : un contexte dans la pile d'exécution (stack frame)

setjmp/longjmp

Passage de paramètres

```
.globl main
main:
    pushl %ebp
    movl  %esp, %ebp
    subl  $8, %esp
    andl  $-16, %esp
    movl  $1, -4(%ebp)
    movl  $1, -8(%ebp)
    subl  $8, %esp

    int main(void){
        int a = 1 ;
        int b = 1 ;

        return 0 ;
    }

    movl  $0, %eax
    leave
    ret
```

www.fil.univ-lille1.fr/~sedoglav/C/Cours09.pdfV48 (19-10-2010)

Pratique du C

Classes d'allocation, contexte et passage de paramètres par la pile

Les classes d'allocation des variables

Effet des mots clef static et extern sur les fonctions

Ordre d'évaluation des paramètres

Fonction à nombre variable de paramètres

Définition : un contexte dans la pile d'exécution (stack frame)

setjmp/longjmp

Passage de paramètres

Pratique du C

Classes d'allocation, contexte et passage de paramètres par la pile

Les classes d'allocation des variables

Effet des mots clef static et extern sur les fonctions

Ordre d'évaluation des paramètres

Fonction à nombre variable de paramètres

Définition : un contexte dans la pile d'exécution (stack frame)

setjmp/longjmp

Passage de paramètres

```
.text
.globl PER
PER:
    pushl %ebp
    movl  %esp, %ebp
    subl  $4, %esp
    movl  8(%ebp), %eax
    movl  %eax, -4(%ebp)

    leave
    ret

setjmp/longjmp
```

void PER(int alpha, int beta){  
 int tmp = alpha ;  
  
}  
  
}

Pratique du C

Classes d'allocation, contexte et passage de paramètres par la pile

Les classes d'allocation des variables

Effet des mots clef static et extern sur les fonctions

Ordre d'évaluation des paramètres

Fonction à nombre variable de paramètres

Définition : un contexte dans la pile d'exécution (stack frame)

setjmp/longjmp

Passage de paramètres

Pratique du C

Classes d'allocation, contexte et passage de paramètres par la pile

Les classes d'allocation des variables

Effet des mots clef static et extern sur les fonctions

Ordre d'évaluation des paramètres

Fonction à nombre variable de paramètres

Définition : un contexte dans la pile d'exécution (stack frame)

setjmp/longjmp

Passage de paramètres

```
.text
.globl PER
PER:
    pushl %ebp
    movl  %esp, %ebp
    subl  $4, %esp
    movl  8(%ebp), %eax
    movl  %eax, -4(%ebp)
    movl  12(%ebp), %eax
    movl  %eax, 8(%ebp)
    movl  -4(%ebp), %eax
    movl  %eax, 12(%ebp)
    leave
    ret

setjmp/longjmp
```

void PER(int alpha, int beta){  
 int tmp = alpha ;  
 alpha = beta ;  
 beta = tmp ;  
  
}  
  
}

Pratique du C

Classes d'allocation, contexte et passage de paramètres par la pile

Les classes d'allocation des variables

Effet des mots clef static et extern sur les fonctions

Ordre d'évaluation des paramètres

Fonction à nombre variable de paramètres

Définition : un contexte dans la pile d'exécution (stack frame)

setjmp/longjmp

Passage de paramètres

Pratique du C

Classes d'allocation, contexte et passage de paramètres par la pile

Les classes d'allocation des variables

Effet des mots clef static et extern sur les fonctions

Ordre d'évaluation des paramètres

Fonction à nombre variable de paramètres

Définition : un contexte dans la pile d'exécution (stack frame)

setjmp/longjmp

Passage de paramètres

```
.text
.globl main
main:
    pushl %ebp
    movl  %esp, %ebp
    subl  $8, %esp
    andl  $-16, %esp
    movl  $1, -4(%ebp)
    movl  $1, -8(%ebp)
    subl  $8, %esp
    leal  -8(%ebp), %eax
    pushl %eax
    leal  -4(%ebp), %eax
    pushl %eax
    call  PER
    addl  $16, %esp
    movl  $0, %eax
    leave
    ret

setjmp/longjmp
```

int main(void){  
  
 int a = 1 ;  
 int b = 1 ;  
 PER(&a,&b) ;  
 return 0;  
  
}  
  
}

Pratique du C

Classes d'allocation, contexte et passage de paramètres par la pile

Les classes d'allocation des variables

Effet des mots clef static et extern sur les fonctions

Ordre d'évaluation des paramètres

Fonction à nombre variable de paramètres

Définition : un contexte dans la pile d'exécution (stack frame)

setjmp/longjmp

Passage de paramètres

```
.text
.globl PER
PER:
    pushl %ebp
    movl  %esp, %ebp
    subl  $4, %esp
    movl  8(%ebp), %eax
    movl  %eax, -4(%ebp)
    movl  12(%ebp), %eax
    movl  %eax, 8(%ebp)

    leave
    ret

setjmp/longjmp
```

void PER(int alpha, int beta){  
 int tmp = alpha ;  
 alpha = beta ;  
  
}  
  
}

## Passage de paramètre par adresse : les adresses sont copiées sur la pile

```
.text
.globl main
main:
    pushl %ebp
    movl  %esp, %ebp
    subl  $8, %esp
    andl  $-16, %esp
    movl  $1, -4(%ebp)
    movl  $1, -8(%ebp)
    subl  $8, %esp

    movl  $0, %eax
    leave
    ret

setjmp/longjmp
```

int main(void){  
  
 int a = 1 ;  
 int b = 1 ;  
  
 return 0;  
  
}  
  
}

```
.text
.globl PER
PER:
    pushl %ebp
    movl  %esp, %ebp
    subl  $4, %esp
    movl  8(%ebp), %eax
    movl  (%eax), %eax
    movl  %eax, -4(%ebp)

    leave
    ret

setjmp/longjmp
```

void PER(int \*alpha, int \*beta){  
 int tmp = \*alpha ;  
  
}  
  
}

Pratique du C  
Classes  
d'allocation,  
contexte et  
passage de  
paramètres par la  
pile

Les classes  
d'allocation des  
variables

Effet des mots clef  
static et extern  
sur les fonctions

Ordre d'évaluation  
des paramètres

Fonction à  
nombre variable  
de paramètres

Définition : un  
contexte dans la  
pile d'exécution  
(stack frame)

setjmp/longjmp

Passage de  
paramètres

V48 (19-10-2010)

Pratique du C  
Classes  
d'allocation,  
contexte et  
passage de  
paramètres par la  
pile

Les classes  
d'allocation des  
variables

Effet des mots clef  
static et extern  
sur les fonctions

Ordre d'évaluation  
des paramètres

Fonction à  
nombre variable  
de paramètres

Définition : un  
contexte dans la  
pile d'exécution  
(stack frame)

setjmp/longjmp

Passage de  
paramètres

V48 (19-10-2010)

Pratique du C  
Classes  
d'allocation,  
contexte et  
passage de  
paramètres par la  
pile

Les classes  
d'allocation des  
variables

Effet des mots clef  
static et extern  
sur les fonctions

Ordre d'évaluation  
des paramètres

Fonction à  
nombre variable  
de paramètres

Définition : un  
contexte dans la  
pile d'exécution  
(stack frame)

setjmp/longjmp

Passage de  
paramètres

V48 (19-10-2010)

.text  
.globl PER  
PER:  
void PER(int \*alpha, int \*beta){  
int tmp = \*alpha ;  
\*alpha = \*beta ;  
}  
pushl %ebp  
movl %esp, %ebp  
subl \$4, %esp  
movl 8(%ebp), %eax  
movl (%eax), %eax  
movl %eax, -4(%ebp)  
movl 8(%ebp), %edx  
movl 12(%ebp), %eax  
movl (%eax), %eax  
movl %eax, (%edx)  
leave  
ret

www.fil.univ-lille1.fr/~sedoglav/C/Cours09.pdfV48 (19-10-2010)

## Passage de paramètre de type structure

.globl main  
main:  
pushl %ebp  
movl %esp, %ebp  
subl \$8, %esp  
andl \$-16, %esp  
movl \$1,-8(%ebp)  
movl \$1,-4(%ebp)  
subl \$8, %esp  
movl \$0, %eax  
leave  
ret  
return 0 ;  
}

www.fil.univ-lille1.fr/~sedoglav/C/Cours09.pdfV48 (19-10-2010)

.text  
.globl UN  
UN:  
pushl %ebp  
movl %esp, %ebp  
subl \$8, %esp  
movl 8(%ebp), %eax  
movl 12(%ebp), %edx  
movl %eax, -8(%ebp)  
movl %edx, -4(%ebp)  
movl \$2, -8(%ebp)  
leave  
ret

www.fil.univ-lille1.fr/~sedoglav/C/Cours09.pdfV48 (19-10-2010)

Pratique du C  
Classes  
d'allocation,  
contexte et  
passage de  
paramètres par la  
pile

Les classes  
d'allocation des  
variables

Effet des mots clef  
static et extern  
sur les fonctions

Ordre d'évaluation  
des paramètres

Fonction à  
nombre variable  
de paramètres

Définition : un  
contexte dans la  
pile d'exécution  
(stack frame)

setjmp/longjmp

Passage de  
paramètres

V48 (19-10-2010)

typedef struct Gauss\_t{  
int re ;  
int im ;  
} Gauss\_t ;  
int main(void){  
struct Gauss\_t var ;  
var.re = 1 ;  
var.im = 1 ;  
UN(var) ;  
return 0 ;  
}

www.fil.univ-lille1.fr/~sedoglav/C/Cours09.pdfV48 (19-10-2010)

## Fonction retournant une structure

typedef struct Gauss\_t{  
int re ;  
int im ;  
} Gauss\_t ;  
int main(void){  
struct Gauss\_t var,res ;  
var.re = 1 ;  
var.im = 1 ;  
res = UN(var) ;  
var.im = res.re ;  
return 0 ;  
}

www.fil.univ-lille1.fr/~sedoglav/C/Cours09.pdfV48 (19-10-2010)

.text  
.globl PER  
PER:  
pushl %ebp  
movl %esp, %ebp  
subl \$4, %esp  
movl 8(%ebp), %eax  
movl (%eax), %eax  
movl %eax, -4(%ebp)  
movl 8(%ebp), %edx  
movl 12(%ebp), %eax  
movl (%eax), %eax  
movl %eax, (%edx)  
movl 12(%ebp), %edx  
movl -4(%ebp), %eax  
movl (%eax), %eax  
movl %eax, (%edx)  
leave  
ret

www.fil.univ-lille1.fr/~sedoglav/C/Cours09.pdfV48 (19-10-2010)

.globl main  
main:  
pushl %ebp  
movl %esp, %ebp  
subl \$8, %esp  
andl \$-16, %esp  
movl \$1,-8(%ebp)  
movl \$1,-4(%ebp)  
subl \$8, %esp  
pushl -4(%ebp)  
pushl -8(%ebp)  
call UN  
addl \$16,  
movl \$0, %eax  
leave  
ret

www.fil.univ-lille1.fr/~sedoglav/C/Cours09.pdfV48 (19-10-2010)

.text  
.globl main  
main:  
pushl %ebp  
movl %esp, %ebp  
subl \$24, %esp  
andl \$-16, %esp  
movl \$1, -8(%ebp)  
movl \$1, -4(%ebp)  
leal -16(%ebp),%eax  
subl \$4, %esp  
pushl -4(%ebp)  
pushl -8(%ebp)  
pushl %eax  
call UN  
addl \$12, %esp  
movl -16(%ebp),%eax  
movl %eax,-4(%ebp)  
movl \$0,%eax  
leave  
ret

www.fil.univ-lille1.fr/~sedoglav/C/Cours09.pdfV48 (19-10-2010)

```
typedef struct Gauss_t{
    int re ;
    int im ;
} Gauss_t ;

struct Gauss_t UN(Gauss_t par){
    par.re = 2 ;
    return par ;
}
```

```

        .text
        .globl UN
UN:
    pushl %ebp

    movl %esp, %ebp
    subl $8, %esp
    movl 8(%ebp), %eax
    movl 12(%ebp), %edx
    movl 16(%ebp), %ecx
    movl %edx, -8(%ebp)
    movl %ecx, -4(%ebp)
    movl $2, -8(%ebp)
    movl -8(%ebp), %edx
    movl -4(%ebp), %ecx
    movl %edx, (%eax)
    movl %ecx, 4(%eax)
    leave
    ret    $4
```