

## Pratique du C Bétisier

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

Semestre 5 — 2013-2014

V-1 (06-12-2012)

```
#define MAX = 10 ;  
int t[MAX], x = MAX ;
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation :
  - ▶ l'exécution :
- ▶ fait ce qu'il devrait :

V84 (06-12-2012)

```
#define add (a,b) (a + b)  
  
int main(void){  
    return add(1,2) ;  
}
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation :
  - ▶ l'exécution :
- ▶ fait ce qu'il devrait :

V84 (06-12-2012)

## Règles du jeu

Les bouts de codes se prêtent aux commentaires.  
On cherche à savoir si ces codes

- ▶ provoquent une erreur à
  - ▶ la compilation,
  - ▶ l'exécution ;
- ▶ font ce qu'il devrait.

V84 (06-12-2012)

```
#define MAX = 10 ;  
int t[MAX], x = MAX ;
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation : oui
  - ▶ l'exécution :
- ▶ fait ce qu'il devrait :

Une macro n'est ni une déclaration ni une initialisation mais  
provoque une substitution textuelle.

V84 (06-12-2012)

```
#define add (a,b) (a + b)  
  
int main(void){  
    return add(1,2) ;  
}
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation : oui
  - ▶ l'exécution :
- ▶ fait ce qu'il devrait :

L'espace est le séparateur entre l'identificateur de macro et la  
chaîne à substituer.

V84 (06-12-2012)

<div>Pratique du C Bétisier</div> <div>Préprocessing</div> <div>Lisibilité du code</div> <div>Efficacité du code</div> <div>Syntaxe</div>	<pre>#define CARRE(a) ((a) * (a)) int main(void){     int x = 2 ;     return CARRE(x++) ; }</pre> <p>Ce bout de code</p> <ul style="list-style-type: none"> <li>▶ provoque une erreur à             <ul style="list-style-type: none"> <li>▶ la compilation :</li> <li>▶ l'exécution :</li> </ul> </li> <li>▶ fait ce qu'il devrait :</li> </ul>	<div>Pratique du C Bétisier</div> <div>Préprocessing</div> <div>Lisibilité du code</div> <div>Efficacité du code</div> <div>Syntaxe</div>	<pre>#define CARRE(a) ((a) * (a)) int main(void){     int x = 2 ;     return CARRE(x++) ; }</pre> <p>Ce bout de code</p> <ul style="list-style-type: none"> <li>▶ provoque une erreur à             <ul style="list-style-type: none"> <li>▶ la compilation : non</li> <li>▶ l'exécution : non</li> </ul> </li> <li>▶ fait ce qu'il devrait : non</li> </ul> <p>Même bien constituée une macro n'immunise pas contre l'effet latéral.</p>
<div>V84 (06-12-2012)</div> <div>Pratique du C Bétisier</div> <div>Préprocessing</div> <div>Lisibilité du code</div> <div>Efficacité du code</div> <div>Syntaxe</div>	<pre>/* On veut retourner 1 */ int main(void){     int a=0, b=0, res ;      if (a)         if (b)  res = !b ;         else  res = !a ;      return res ; }</pre> <p>Ce bout de code</p> <ul style="list-style-type: none"> <li>▶ provoque une erreur à             <ul style="list-style-type: none"> <li>▶ la compilation :</li> <li>▶ l'exécution :</li> </ul> </li> <li>▶ fait ce qu'il devrait :</li> </ul>	<div>V84 (06-12-2012)</div> <div>Pratique du C Bétisier</div> <div>Préprocessing</div> <div>Lisibilité du code</div> <div>Efficacité du code</div> <div>Syntaxe</div>	<pre>/* On veut retourner 1 */ int main(void){     int a=0, b=0, res ;      if (a)         if (b)  res = !b ;         else  res = !a ;      return res ; }</pre> <p>Ce bout de code</p> <ul style="list-style-type: none"> <li>▶ provoque une erreur à             <ul style="list-style-type: none"> <li>▶ la compilation : non</li> <li>▶ l'exécution : oui</li> </ul> </li> <li>▶ fait ce qu'il devrait :</li> </ul> <p>Il est difficile de savoir à quel if se vouer, alors autant utiliser des blocs.</p>
<div>V84 (06-12-2012)</div> <div>Pratique du C Bétisier</div> <div>Préprocessing</div> <div>Lisibilité du code</div> <div>Efficacité du code</div> <div>Syntaxe</div>	<pre>int main(void){     char c ;     while (c = getchar() != EOF) putchar(c) ;     return 0 ; }</pre> <p>Ce bout de code</p> <ul style="list-style-type: none"> <li>▶ provoque une erreur à             <ul style="list-style-type: none"> <li>▶ la compilation :</li> <li>▶ l'exécution :</li> </ul> </li> <li>▶ fait ce qu'il devrait :</li> </ul>	<div>V84 (06-12-2012)</div> <div>Pratique du C Bétisier</div> <div>Préprocessing</div> <div>Lisibilité du code</div> <div>Efficacité du code</div> <div>Syntaxe</div>	<pre>int main(void){     char c ;     while (c = getchar() != EOF) putchar(c) ;     return 0 ; }</pre> <p>Ce bout de code</p> <ul style="list-style-type: none"> <li>▶ provoque une erreur à             <ul style="list-style-type: none"> <li>▶ la compilation : non</li> <li>▶ l'exécution : non</li> </ul> </li> <li>▶ fait ce qu'il devrait : oui</li> </ul> <p>Il faut rendre lisible le code plutôt que de compter sur la priorité entre les opérateurs.</p>

Pratique du C  
Bétisier

Préprocessing

Lisibilité du code

Efficacité du code

Syntaxe

```
/* On veut parcourir une cha\^ine de caract\`eres */
int
main
(void)
{ int i ;
  char * bibi="La vie est belle" ;
  for(i=0;i<strlen(bibi);i++);
  return 0 ;
}
```

Ce bout de code

▶

provoque une erreur à

▶

la compilation :

▶

l'exécution :

▶

fait ce qu'il devrait :

V84 (06-12-2012)

www.fil.univ-lille1.fr/~sedoglav/C/Cours11.pdf

Pratique du C  
Bétisier

Préprocessing

Lisibilité du code

Efficacité du code

Syntaxe

```
/* On veut parcourir une cha\^ine de caract\`eres */
int
main
(void)
{ int i ;
  char * bibi="La vie est belle" ;
  for(i=0;i<strlen(bibi);i++);
  return 0 ;
}
```

Ce bout de code

▶

provoque une erreur à

▶

la compilation : non

▶

l'exécution : non

▶

fait ce qu'il devrait : oui

C'est un parcours en  $n^2$ !!!!!! À éviter impérativement.

Utilisez les fonctions de la librairie standard à loisir après les avoir comprises (i.e. après votre cours de pratique des systèmes).

V84 (06-12-2012)

www.fil.univ-lille1.fr/~sedoglav/C/Cours11.pdf

Pratique du C  
Bétisier

Préprocessing

Lisibilité du code

Efficacité du code

Syntaxe

```
int main(void){
  int a=1,b=1 ;
  if(a=0)
    printf("Attention z\\\`ero") ;
  else b /= a ;
  return b ;
}
```

Ce bout de code

▶

provoque une erreur à

▶

la compilation :

▶

l'exécution :

▶

fait ce qu'il devrait :

V84 (06-12-2012)

www.fil.univ-lille1.fr/~sedoglav/C/Cours11.pdf

Pratique du C  
Bétisier

Préprocessing

Lisibilité du code

Efficacité du code

Syntaxe

```
int main(void){
  int a=1,b=1 ;
  if(a=0)
    printf("Attention z\\\`ero") ;
  else b /= a ;
  return b ;
}
```

Ce bout de code

▶

provoque une erreur à

▶

la compilation : non

▶

l'exécution : non

▶

fait ce qu'il devrait : non

L'affectation est un opérateur et non une instruction.

V84 (06-12-2012)

www.fil.univ-lille1.fr/~sedoglav/C/Cours11.pdf

Pratique du C  
Bétisier

Préprocessing

Lisibilité du code

Efficacité du code

Syntaxe

```
/* on veut retourner 0 */
int main(void){
  char a=1,b=0 ;
  switch(a){
    case 1 : a = b ;
    default : return 1 ;
  }
  return a ;
}
```

Ce bout de code

▶

provoque une erreur à

▶

la compilation :

▶

l'exécution :

▶

fait ce qu'il devrait :

V84 (06-12-2012)

www.fil.univ-lille1.fr/~sedoglav/C/Cours11.pdf

Pratique du C  
Bétisier

Préprocessing

Lisibilité du code

Efficacité du code

Syntaxe

```
/* on veut retourner 0 */
int main(void){
  char a=1,b=0 ;
  switch(a){
    case 1 : a = b ;
    default : return 1 ;
  }
  return a ;
}
```

Ce bout de code

▶

provoque une erreur à

▶

la compilation : non

▶

l'exécution : non

▶

fait ce qu'il devrait : non

L'existence du goto et l'usage du break induisent cette erreur difficilement détectable (sans l'aide des avertissements du compilateur).

V84 (06-12-2012)

www.fil.univ-lille1.fr/~sedoglav/C/Cours11.pdf

Pratique du C  
Bétisier

Préprocessing  
Lisibilité du code  
Efficacité du code  
Syntaxe

```
int tab[12] = {
    1, 2, 3, 4,
    5, 6, 7, 8,
    9,10,11,12
} ;

int main(void){ /* on veut retourner 3 */
    int i=2,j=2 ;
    return tab[i++,j++] ;
}
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation :
  - ▶ l'exécution :
- ▶ fait ce qu'il devrait :

V84 (06-12-2012)

www.fil.univ-lille1.fr/~sedoglav/C/Cours11.pdf

Pratique du C  
Bétisier

Préprocessing  
Lisibilité du code  
Efficacité du code  
Syntaxe

```
int tab[12] = {
    1, 2, 3, 4,
    5, 6, 7, 8,
    9,10,11,12
} ;

int main(void){ /* on veut retourner 3 */
    int i=2,j=2 ;
    return tab[i++,j++] ;
}
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation : non
  - ▶ l'exécution : non
- ▶ fait ce qu'il devrait : oui

La virgule est l'opérateur qui délivre comme résultat l'opérande droit après avoir évalué l'opérande gauche.

V84 (06-12-2012)

www.fil.univ-lille1.fr/~sedoglav/C/Cours11.pdf

Pratique du C  
Bétisier

Préprocessing  
Lisibilité du code  
Efficacité du code  
Syntaxe

```
/* On veut retourner 2 au shell */
int main(void){
    return fct(2) ;
}

int fct(int i){
    return i ;
}
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation :
  - ▶ l'exécution :
- ▶ fait ce qu'il devrait :

V84 (06-12-2012)

www.fil.univ-lille1.fr/~sedoglav/C/Cours11.pdf

Pratique du C  
Bétisier

Préprocessing  
Lisibilité du code  
Efficacité du code  
Syntaxe

```
/* On veut retourner 2 au shell */
int main(void){
    return fct(2) ;
}

int fct(int i){
    return i ;
}
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation : oui
  - ▶ l'exécution :
- ▶ fait ce qu'il devrait :

La fonction fct devrait être déclarée par un prototype avant la fonction principale.

V84 (06-12-2012)

www.fil.univ-lille1.fr/~sedoglav/C/Cours11.pdf

Pratique du C  
Bétisier

Préprocessing  
Lisibilité du code  
Efficacité du code  
Syntaxe

```
int pair(int i){
    if (i)
        return impair(i-1) ;
    return 1 ;
}

int impair (int i){
    if (i)
        return pair(i-1) ;
    return 0 ;
}
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation :
  - ▶ l'exécution :
- ▶ fait ce qu'il devrait :

V84 (06-12-2012)

www.fil.univ-lille1.fr/~sedoglav/C/Cours11.pdf

Pratique du C  
Bétisier

Préprocessing  
Lisibilité du code  
Efficacité du code  
Syntaxe

```
int pair(int i){
    if (i)
        return impair(i-1) ;
    return 1 ;
}

int impair (int i){
    if (i)
        return pair(i-1) ;
    return 0 ;
}
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation : oui
  - ▶ l'exécution :
- ▶ fait ce qu'il devrait :

Encore une fois, un prototype permet de fournir au compilateur la déclaration qui lui manque.

V84 (06-12-2012)

www.fil.univ-lille1.fr/~sedoglav/C/Cours11.pdf

Pratique du C  
Bétisier

Préprocessing  
Lisibilité du code  
Efficacité du code  
Syntaxe

```
typedef int fct_t (int) ;

fct_t fct ;

int main(void){
    return fct(2) ;
}

int fct(int i){
    return i ;
}
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation :
  - ▶ l'exécution :
- ▶ fait ce qu'il devrait :

V84 (06-12-2012)

Pratique du C  
Bétisier

Préprocessing  
Lisibilité du code  
Efficacité du code  
Syntaxe

```
int fct(int i){
    return i ;
}

int main(void){
    fct(2) ;
    return 0 ;
}
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation :
  - ▶ l'exécution :
- ▶ fait ce qu'il devrait :

Pratique du C  
Bétisier

Préprocessing  
Lisibilité du code  
Efficacité du code  
Syntaxe

```
typedef int fct_t (int) ;

fct_t fct ;

int main(void){
    return fct(2) ;
}

int fct(int i){
    return i ;
}
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation : non
  - ▶ l'exécution : non
- ▶ fait ce qu'il devrait : oui

Un prototype de fonction est une déclaration.

V84 (06-12-2012)

Pratique du C  
Bétisier

Préprocessing  
Lisibilité du code  
Efficacité du code  
Syntaxe

```
int fct(int i){
    return i ;
}

int main(void){
    fct(2) ;
    return 0 ;
}
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation : non
  - ▶ l'exécution : non
- ▶ fait ce qu'il devrait : oui

Même si la fonction fct retourne une valeur, rien ne nous force à la récupérer.

V84 (06-12-2012)

Pratique du C  
Bétisier

Préprocessing  
Lisibilité du code  
Efficacité du code  
Syntaxe

```
int MaVar = 1 ;

int main(void){
    return ++Mavar ;
}
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation :
  - ▶ l'exécution :
- ▶ fait ce qu'il devrait :

V84 (06-12-2012)

Pratique du C  
Bétisier

Préprocessing  
Lisibilité du code  
Efficacité du code  
Syntaxe

```
int MaVar = 1 ;

int main(void){
    return ++Mavar ;
}
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation :
  - ▶ l'exécution :
- ▶ fait ce qu'il devrait :

V84 (06-12-2012)

Pratique du C  
Bétisier

Préprocessing  
Lisibilité du code  
Efficacité du code  
Syntaxe

```
int MaVar = 1 ;

int main(void){
    return ++Mavar ;
}
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation : oui
  - ▶ l'exécution :
- ▶ fait ce qu'il devrait :

Les identificateurs sont sensibles à la casse.

<div>Pratique du C Bétisier</div> <div>Préprocessing</div> <div>Lisibilité du code</div> <div>Efficacité du code</div> <div>Syntaxe</div>	<pre>int var(void){ return 2;} int main(void){     int a = 10, var = 1, cinq = a/var ;     return cinq ; }</pre> <p>Ce bout de code</p> <ul style="list-style-type: none"><li>▶ provoque une erreur à<ul style="list-style-type: none"><li>▶ la compilation :</li><li>▶ l'exécution :</li></ul></li><li>▶ fait ce qu'il devrait :</li></ul>	<div>Pratique du C Bétisier</div> <div>Préprocessing</div> <div>Lisibilité du code</div> <div>Efficacité du code</div> <div>Syntaxe</div>	<pre>int var(void){ return 2;} int main(void){     int a = 10, var = 1, cinq = a/var ;     return cinq ; }</pre> <p>Ce bout de code</p> <ul style="list-style-type: none"><li>▶ provoque une erreur à<ul style="list-style-type: none"><li>▶ la compilation : non</li><li>▶ l'exécution : non</li></ul></li><li>▶ fait ce qu'il devrait : non</li></ul> <p>La porté des identificateurs n'est pas un vain mot.</p>
V84 (06-12-2012)	<div>Pratique du C Bétisier</div> <div>Préprocessing</div> <div>Lisibilité du code</div> <div>Efficacité du code</div> <div>Syntaxe</div>	<div>Pratique du C Bétisier</div> <div>Préprocessing</div> <div>Lisibilité du code</div> <div>Efficacité du code</div> <div>Syntaxe</div>	V84 (06-12-2012)
<div>Pratique du C Bétisier</div> <div>Préprocessing</div> <div>Lisibilité du code</div> <div>Efficacité du code</div> <div>Syntaxe</div>	<pre>int deux(void){ return 2 ; }  int main(void){     int (*deux) = deux ;     return deux ; }</pre> <p>Ce bout de code</p> <ul style="list-style-type: none"><li>▶ provoque une erreur à<ul style="list-style-type: none"><li>▶ la compilation :</li><li>▶ l'exécution :</li></ul></li><li>▶ fait ce qu'il devrait :</li></ul>	<div>Pratique du C Bétisier</div> <div>Préprocessing</div> <div>Lisibilité du code</div> <div>Efficacité du code</div> <div>Syntaxe</div>	<pre>int deux(void){ return 2 ; }  int main(void){     int (*deux) = deux ;     return deux ; }</pre> <p>Ce bout de code</p> <ul style="list-style-type: none"><li>▶ provoque une erreur à<ul style="list-style-type: none"><li>▶ la compilation : non</li><li>▶ l'exécution : non</li></ul></li><li>▶ fait ce qu'il devrait : non</li></ul> <p>Ce code retourne une adresse.</p>
V84 (06-12-2012)	<div>Pratique du C Bétisier</div> <div>Préprocessing</div> <div>Lisibilité du code</div> <div>Efficacité du code</div> <div>Syntaxe</div>	<div>Pratique du C Bétisier</div> <div>Préprocessing</div> <div>Lisibilité du code</div> <div>Efficacité du code</div> <div>Syntaxe</div>	V84 (06-12-2012)
<div>Pratique du C Bétisier</div> <div>Préprocessing</div> <div>Lisibilité du code</div> <div>Efficacité du code</div> <div>Syntaxe</div>	<p>On souhaite retourner la somme de deux entiers saisis au clavier.</p> <pre>int main(void){     int a=0, b=0 ;     scanf("%d%d",a,b) ;     return a+b ; }</pre> <p>Ce bout de code</p> <ul style="list-style-type: none"><li>▶ provoque une erreur à<ul style="list-style-type: none"><li>▶ la compilation :</li><li>▶ l'exécution :</li></ul></li><li>▶ fait ce qu'il devrait :</li></ul>	<div>Pratique du C Bétisier</div> <div>Préprocessing</div> <div>Lisibilité du code</div> <div>Efficacité du code</div> <div>Syntaxe</div>	<p>On souhaite retourner la somme de deux entiers saisis au clavier.</p> <pre>int main(void){     int a=0, b=0 ;     scanf("%d%d",a,b) ;     return a+b ; }</pre> <p>Ce bout de code</p> <ul style="list-style-type: none"><li>▶ provoque une erreur à<ul style="list-style-type: none"><li>▶ la compilation : non</li><li>▶ l'exécution : non</li></ul></li><li>▶ fait ce qu'il devrait : non</li></ul> <p>Le passage de paramètre par référence permet d'exporter de l'information d'une fonction.</p>
V84 (06-12-2012)	<div>Pratique du C Bétisier</div> <div>Préprocessing</div> <div>Lisibilité du code</div> <div>Efficacité du code</div> <div>Syntaxe</div>	<div>Pratique du C Bétisier</div> <div>Préprocessing</div> <div>Lisibilité du code</div> <div>Efficacité du code</div> <div>Syntaxe</div>	V84 (06-12-2012)

- Préprocessing
- Lisibilité du code
- Efficacité du code
- Syntaxe**

```
int SommeDes10Premiers(int tab[10]) {
    int i, res = 0 ;
    for(i=1;i<11;i++)
        res += tab[i] ;
    return res ;
}
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation :
  - ▶ l'exécution :
- ▶ fait ce qu'il devrait :

- Préprocessing
- Lisibilité du code
- Efficacité du code
- Syntaxe**

```
int main(void){
    int zero = 0 ;
    int tab[5] = {1,2,3,4,10} ;
    zero = 0[tab] + *(tab+1) + *(2+tab) + tab[3] - 4[tab];
    return zero ;
}
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation :
  - ▶ l'exécution :
- ▶ fait ce qu'il devrait :

- Préprocessing
- Lisibilité du code
- Efficacité du code
- Syntaxe**

```
#include<stdio.h>
int main(void){
    int i = 7, a[5]; char c = 0 ;
    /* qu'est qui est affich\`e */
    printf( "%d", i++ * i++ );
    /* comment est modifi\`e ce tableau */
    i = 1; a[i] = i++;
    /* ce code peut il s'arr\`eter */
    do { c = getchar(); } while(c != EOF) ;
    return 0 ;
}
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation :
  - ▶ l'exécution :
- ▶ fait ce qu'il devrait :

- Préprocessing
- Lisibilité du code
- Efficacité du code
- Syntaxe**

```
int SommeDes10Premiers(int tab[10]) {
    int i, res = 0 ;
    for(i=1;i<11;i++)
        res += tab[i] ;
    return res ;
}
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation : non
  - ▶ l'exécution : non
- ▶ fait ce qu'il devrait : non

Les indices de tableaux commencent à 0 et rien ne vous empêche de passer outre.

- Préprocessing
- Lisibilité du code
- Efficacité du code
- Syntaxe**

```
int main(void){
    int zero = 0 ;
    int tab[5] = {1,2,3,4,10} ;
    zero = 0[tab] + *(tab+1) + *(2+tab) + tab[3] - 4[tab];
    return zero ;
}
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation : non
  - ▶ l'exécution : non
- ▶ fait ce qu'il devrait : oui

La notation `tab[1]` est équivalente à `*(tab+1)` ce qui permet ce code surprenant.

- Préprocessing
- Lisibilité du code
- Efficacité du code
- Syntaxe**

```
#include<stdio.h>
int main(void){
    int i = 7, a[5]; char c = 0 ;
    /* qu'est qui est affich\`e */
    printf( "%d", i++ * i++ );
    /* comment est modifi\`e ce tableau */
    i = 1; a[i] = i++;
    /* ce code peut il s'arr\`eter */
    do { c = getchar(); } while(c != EOF) ;
    return 0 ;
}
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation : non
  - ▶ l'exécution : non
- ▶ fait ce qu'il devrait : ???

Si on ne lit pas la documentation du compilateur ou l'assembleur, il existe des questions sans réponses i.e. la norme ne spécifie pas tout.

Pratique du C  
Bétisier

Préprocessing  
Lisibilité du code  
Efficacité du code  
Syntaxe

```
/* ce code retourne 100 */
int main(void){
    i=0 ; j=1 ;
    while(i<100) ;
        i=i+j ;
    return i ;
}
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation :
  - ▶ l'exécution :
- ▶ fait ce qu'il devrait :

V84 (06-12-2012)

Pratique du C  
Bétisier

Préprocessing  
Lisibilité du code  
Efficacité du code  
Syntaxe

```
char foo[10] = '\0' ;
char bar[10] = "\0" ;

int * ptr1, * ptr2 ;
int * pptr1, pptr2 ;
pptr1 = ptr1 ;
pptr2 = ptr2 ;
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation :
  - ▶ l'exécution :
- ▶ fait ce qu'il devrait :

Pratique du C  
Bétisier

Préprocessing  
Lisibilité du code  
Efficacité du code  
Syntaxe

```
/* ce code retourne 100 */
int main(void){
    i=0 ; j=1 ;
    while(i<100) ;
        i=i+j ;
    return i ;
}
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation : non
  - ▶ l'exécution : non
- ▶ fait ce qu'il devrait : non

Attention aux trivialités.

V84 (06-12-2012)

Pratique du C  
Bétisier

Préprocessing  
Lisibilité du code  
Efficacité du code  
Syntaxe

```
char foo[10] = '\0' ;
char bar[10] = "\0" ;

int * ptr1, * ptr2 ;
int * pptr1, pptr2 ;
pptr1 = ptr1 ;
pptr2 = ptr2 ;
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation :
  - ▶ l'exécution :
- ▶ fait ce qu'il devrait :

De petites différences produisent de grands effets.

Pratique du C  
Bétisier

Préprocessing  
Lisibilité du code  
Efficacité du code  
Syntaxe

```
int *a[10] ;
int (*b)[10] ;
b[9]=a[0] ;
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation :
  - ▶ l'exécution :
- ▶ fait ce qu'il devrait :

V84 (06-12-2012)

Pratique du C  
Bétisier

Préprocessing  
Lisibilité du code  
Efficacité du code  
Syntaxe

```
int *a[10] ;
int (*b)[10] ;
b[9]=a[0] ;
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation : oui
  - ▶ l'exécution :
- ▶ fait ce qu'il devrait :

La compilation signale error : incompatible types in assignment. En effet, le pointeur d'entier a[0] pourrait "avoir" plus de 10 cellules.

Pratique du C  
Bétisier

Préprocessing  
Lisibilité du code  
Efficacité du code  
Syntaxe

```
int *a[10] ;
int (*b)[10] ;
b[9]=a[0] ;
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation : oui
  - ▶ l'exécution :
- ▶ fait ce qu'il devrait :

La compilation signale error : incompatible types in assignment. En effet, le pointeur d'entier a[0] pourrait "avoir" plus de 10 cellules.

V84 (06-12-2012)

Pratique du C  
Bétisier

Préprocessing  
Lisibilité du code  
Efficacité du code  
Syntaxe

```
int *a[10] ;
int (*b)[10] ;
b[9]=a[0] ;
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation : oui
  - ▶ l'exécution :
- ▶ fait ce qu'il devrait :

La compilation signale error : incompatible types in assignment. En effet, le pointeur d'entier a[0] pourrait "avoir" plus de 10 cellules.



```
int main(void){
/* Emboitementement de /* commentaire */ */
int a,b,c = 3;
int *pointer = &c ;
/*divise c par lui m^eme */
a=c/*pointer ;
b=c /*met b \a 3*/ ;
return 0 ;
}
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation :
  - ▶ l'exécution :
- ▶ fait ce qu'il devrait :

```
#include<stdio.h>
char *foo(void){
    char str[31]="Pourquoi vais-je disparaître?" ;
    return str ;
}
```

```
int main(void){
    printf("%d %d %s\n",1,2,foo()) ;
    return 0;
}
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation :
  - ▶ l'exécution :
- ▶ fait ce qu'il devrait :

```
int
main
(void)
{
    char *str = "stringX" ;
    str[6]='Y' ;
    return 0;
}
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation :
  - ▶ l'exécution :
- ▶ fait ce qu'il devrait :

www.fil.univ-lille1.fr/~sedoglav/C/Cours11.pdf

```
int main(void){
/* Emboitement de /* commentaire */ */
int a,b,c = 3;
int *pointer = &c ;
/*divise c par lui m\^eme */
a=c/*pointer ;
b=c /*met b \`a 3*/ ;
return 0 ;
}
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation : non
  - ▶ l'exécution : non
- ▶ fait ce qu'il devrait : non

Attention aux commentaires.

```
#include<stdio.h>
char *foo(void){
    char str[31]="Pourquoi vais-je disparaître?" ;
    return str ;
}
```

```
int main(void){
    printf("%d %d %s\n",1,2,foo()) ;
    return 0;
}
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation : non
  - ▶ l'exécution : non
- ▶ fait ce qu'il devrait : non

Le compilateur indique que warning: function returns address of local variable.

```
int
main
(void)
{
    char *str = "stringX" ;
    str[6]='Y' ;
    return 0;
}
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation : non
  - ▶ l'exécution : non
- ▶ fait ce qu'il devrait : oui

La chaîne "stringX" est stockée dans un segment en lecture seule (.rodata) et la tentative d'accès se solde par un `segmentation fault`.

www.fil.univ-lille1.fr/~sedoglav/C/Cours11.pdf

Pratique du C  
Bétisier

Préprocessing  
Lisibilité du code  
Efficacité du code  
Syntaxe

```
#include<stdio.h>
struct tab_s{ int tab[2] ; } ;

struct tab_s foo(void){
    struct tab_s res ;
    res.tab[0] = 1 ; res.tab[1] = 2 ;
    return res ;
}

int main(void){
    struct tab_s tmp = foo() ;
    int zero = 2*tmp.tab[0] - tmp.tab[1] ;
    return zero ;
}
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation :
  - ▶ l'exécution :
- ▶ fait ce qu'il devrait :

V84 (06-12-2012)

www.fil.univ-lille1.fr/~sedoglav/C/Cours11.pdf

Pratique du C  
Bétisier

Préprocessing  
Lisibilité du code  
Efficacité du code  
Syntaxe

```
#include<stdio.h>
struct tab_s{ int tab[2] ; } ;

struct tab_s foo(void){
    struct tab_s res ;
    res.tab[0] = 1 ; res.tab[1] = 2 ;
    return res ;
}

int main(void){
    struct tab_s tmp = foo() ;
    int zero = 2*tmp.tab[0] - tmp.tab[1] ;
    return zero ;
}
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation : non
  - ▶ l'exécution : non
- ▶ fait ce qu'il devrait : oui

On ne peut affecter un tableau à un autre, c'est possible pour les structures.

V84 (06-12-2012)

www.fil.univ-lille1.fr/~sedoglav/C/Cours11.pdf

Pratique du C  
Bétisier

Préprocessing  
Lisibilité du code  
Efficacité du code  
Syntaxe

Ce code retourne la taille de la chaîne de caractères

```
int main(void){
    char *ch1 = "Hello world";
    return sizeof(ch1) ;
}
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation :
  - ▶ l'exécution :
- ▶ fait ce qu'il devrait :

V84 (06-12-2012)

www.fil.univ-lille1.fr/~sedoglav/C/Cours11.pdf

Pratique du C  
Bétisier

Préprocessing  
Lisibilité du code  
Efficacité du code  
Syntaxe

Ce code retourne la taille de la chaîne de caractères

```
int main(void){
    char *ch1 = "Hello world";
    return sizeof(ch1) ;
}
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation : non
  - ▶ l'exécution : non
- ▶ fait ce qu'il devrait : non

N'utilisez sizeof que sur des types.

V84 (06-12-2012)

www.fil.univ-lille1.fr/~sedoglav/C/Cours11.pdf

Pratique du C  
Bétisier

Préprocessing  
Lisibilité du code  
Efficacité du code  
Syntaxe

Une implantation de memset

```
void *memset(void *b, int c, int len)
{
    int i ;
    for(i=0;i<len;i++)
        b[i]=c ;
    return b ;
}
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation :
  - ▶ l'exécution :
- ▶ fait ce qu'il devrait :

V84 (06-12-2012)

www.fil.univ-lille1.fr/~sedoglav/C/Cours11.pdf

Pratique du C  
Bétisier

Préprocessing  
Lisibilité du code  
Efficacité du code  
Syntaxe

Une implantation de memset

```
void *memset(void *b, int c, int len)
{
    int i ;
    for(i=0;i<len;i++)
        b[i]=c ;
    return b ;
}
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation : non
  - ▶ l'exécution :
- ▶ fait ce qu'il devrait :

Ni arithmétique ni déréférencement de pointeur void .

V84 (06-12-2012)

www.fil.univ-lille1.fr/~sedoglav/C/Cours11.pdf

Pratique du C  
Bétisier

Préprocessing  
Lisibilité du code  
Efficacité du code  
Syntaxe

```
int main(void){
    int *p ;
    p = malloc(10*sizeof (int));
    p[10] = 666 ;
    return 0 ;
}
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation :
  - ▶ l'exécution :
- ▶ fait ce qu'il devrait :

V84 (06-12-2012)

www.fil.univ-lille1.fr/~sedoglav/C/Cours11.pdf

Pratique du C  
Bétisier

Préprocessing  
Lisibilité du code  
Efficacité du code  
Syntaxe

```
int main(void){
    int *p ;
    p = malloc(10*sizeof (int));
    p[10] = 666 ;
    return 0 ;
}
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation : non
  - ▶ l'exécution : non
- ▶ fait ce qu'il devrait : non

Affectation hors limite.

V84 (06-12-2012)

www.fil.univ-lille1.fr/~sedoglav/C/Cours11.pdf

Pratique du C  
Bétisier

Préprocessing  
Lisibilité du code  
Efficacité du code  
Syntaxe

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

/* strtod, strttof, strtold - convert
ASCII string to floating point number
float strttof(const char *, char **); */

int main(int argc, char **argv){
    printf("%f\n", strttof(argv[1], NULL)) ;
    return 0 ;
}
```

Si on compile avec l'option ansi, le prototype de la fonction strttof — n'étant pas dans cette norme et étant à l'intérieur d'une directive conditionnelle — n'est pas pris en compte. Sans prototype, la valeur de retour de la fonction est supposée être un entier machine et le résultat n'est pas codé comme un flottant et donc faux.

V84 (06-12-2012)

www.fil.univ-lille1.fr/~sedoglav/C/Cours11.pdf

Pratique du C  
Bétisier

Préprocessing  
Lisibilité du code  
Efficacité du code  
Syntaxe

```
int main(void) {
    int i ;
    int tampon[5] ;

    for(i=0;i<666;i++)
        tampon[i]=2 ;
    return 0;
}
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation :
  - ▶ l'exécution :
- ▶ fait ce qu'il devrait :

V84 (06-12-2012)

www.fil.univ-lille1.fr/~sedoglav/C/Cours11.pdf

Pratique du C  
Bétisier

Préprocessing  
Lisibilité du code  
Efficacité du code  
Syntaxe

```
int main(void) {
    int i ;
    int tampon[5] ;

    for(i=0;i<666;i++)
        tampon[i]=2 ;
    return 0;
}
```

Ce bout de code

- ▶ provoque une erreur à
  - ▶ la compilation : non
  - ▶ l'exécution : non
- ▶ fait ce qu'il devrait : non

Cela provoque une boucle infinie!!!

V84 (06-12-2012)

www.fil.univ-lille1.fr/~sedoglav/C/Cours11.pdf

Pratique du C  
Bétisier

Préprocessing  
Lisibilité du code  
Efficacité du code  
Syntaxe

Le code

```
int main(void){
    int tab[10] ;
    int i = 0 ;
    for(;i<300;i++)
        tab[i] = i ;
    return 0 ;
}
```

donne l'exécution suivante :

```
espoir % gcc code.c
espoir % a.out
Segmentation fault
```

V84 (06-12-2012)

www.fil.univ-lille1.fr/~sedoglav/C/Cours11.pdf

