

# S: EXEMPLE 1

```
length)
```

```
+ ) {  
) {  
) {  
];
```

# PROBLÈMES:

```
void tri_decroissant(int *tab, int len  
{  
    int i, j;  
    for (i = length - 1; i > 0; i++) {  
        for (j = 0; j < i - 1; j++) {  
            if (tab[j] < tab[j + 1]) {  
                int temp = tab[j];  
                tab[j] = tab[j + 1];  
                tab[j + 1] = temp;  
            }  
        }  
    }  
}
```



## DE FONCTION

ne fonction ?

teur de fonction ?

ction référencée ?

de pointeurs de fonction ?

## RAPPELS SUR LE

- Toute variable est stockée soit  
segment de données (.data ou
- Ce stockage lui confère une ad  
stockée dans une variable de t

connaître son "type", comme pour

```
void * ptr_foo = &(foo);  
void * ptr_foo2 = foo;  
printf("@foo:%p - %p\n", ptr_foo,  
return 0;  
}
```

pond à un pointeur statique

de la fonction conduit à son  
optionel, mais recommandé

## RÉCUPÉRATION DE L'ADRE

- Soucis pointeur générique
  - bien que gérée par certains compilateurs, elle n'est pas portable
  - il n'est pas possible de déréf

ur de fonction s'effectue ainsi:

```
es_fct);  
pes des paramètres de la  
pointer
```

```
void (*ptr_1)(void),  
void (*ptr_2)(int);  
void (*ptr_3)(int, char *, size_t)  
ptr_1=&(foo_1);  
ptr_2=&(foo_2);  
ptr_3=&(foo_3);  
printf("%p - %p - %p\n",ptr_1,ptr_  
return 0;  
}
```

## MPLES

e fonctions ne retournant un

```
t 1){...}
```

## RAPPEL: APPEL

- L'appel d'une fonction s'effectue son nom suivi entre parenthèses
- Le compilateur va remplacer c l'exécution du code correspon

```

void foo_1(void){ printf("foo_1\n"); }
void foo_2(int i){ printf("foo_2:%d\n", i); }
int foo_3(int i, char * s, size_t l){
    printf("foo_3:%d,%s,%d\n", i, s, l);
    return 1;
}
int main(void){
    void (*ptr_1)(void)=&(foo_1);
    void (*ptr_2)(int)=&(foo_2);
    int (*ptr_3)(int, char *, size_t)=
    int return_value;
    (*ptr_1)();
    (*ptr_2)(2);
    return_value = (*ptr_3)(2, "Hello",
    printf("Return value: %d\n", return_value);
    return 0;
}

```

## TABLEAUX DE FONCTIONS

Une fonction est une adresse, il est possible de stocker des adresses dans un tableau

On peut déclarer un tableau d'un type "simple" comme un tableau de pointeurs. La déclaration d'un tableau de pointeurs est la même que celle d'un tableau de variables.

## TABLEAU DE POINTEURS

- Ces tableaux sont soumis aux mêmes règles que les tableaux "standards":
  - l'indice commence à 0
  - un tableau ne connaît pas sa taille

```
foo(l_args_foo))  
es_fct);
```

paramètre une liste  
foo) et retourne un pointeur  
prototype est type\_ret\_fct

## D'UN TYPEDEF

déclarant un nouveau type

```
int main(void){  
    void (*ptr_1)(void)=foo_3(1);  
    void (*ptr_2)(char *)=foo_4(2);  
    (*ptr_2)("Hello");  
    (*ptr_1)();  
    return 0;  
}
```

Que ce code a

## POINTEUR DE FONCTION

- Pour passer un pointeur de fonction  
de déclarer le paramètre comme

```
void foo_1(int i, char * s){  
    printf("%s : %d\n", s, i);  
}
```

sont l'addition, la soustraction, la

Un calcul est composé d'un premier  
opération et d'un second opérateur  
résultat

## UTILITÉ: EXEMPLE

nt implémentées comme suit et  
de pointeurs de fonctions

b){

## RETOUR SUR L'UT

- Le choix de l'opération

```
double (* selectOperation(char choice)
int i=-1;
switch(choice){
    case '+': i=0; break;
    case '-': i=1; break;
    case 'x': i=2; break;
```

End),  
;

type de fonction correspondant

- Les erreurs sont faciles et définies, nécessitent une grande vigilance