Langage C

ASR2 - Système

(deuxième séance)

12 avril 2013

Upper case vs. Lower case

```
#include <stdio.h>                                  // printf() et scanf()
#include <stdlib.h>                                        // EXIT_SUCCESS
#include <string.h>                                           // strlen()
#include <ctype.h>                                 // tolower() et toupper()

int main (int argc, char *argv[]){
   if (argc != 2){                                   // test ultra minimal..
      printf("Usage: %s <upper|lower>\n", argv[0]);
      return EXIT_FAILURE;
   }









}
```

Upper case vs. Lower case

```
#include <stdio.h>                                  // printf() et scanf()
#include <stdlib.h>                                        // EXIT_SUCCESS
#include <string.h>                                           // strlen()
#include <ctype.h>                                 // tolower() et toupper()

int main (int argc, char *argv[]){
   if (argc != 2){                                   // test ultra minimal..
      printf("Usage: %s <upper|lower>\n", argv[0]);
      return EXIT_FAILURE;
   }
   char chaine[100];
   while(1){
      scanf("%s", chaine);









      printf("%s\n", chaine);
   }
}
```

Upper case vs. Lower case

```
#include <stdio.h>                                  // printf() et scanf()
#include <stdlib.h>                                      // EXIT_SUCCESS
#include <string.h>                                         // strlen()
#include <ctype.h>                                // tolower() et toupper()

int main (int argc, char *argv[]){
   if (argc != 2){                                  // test ultra minimal..
      printf("Usage: %s <upper|lower>\n", argv[0]);
      return EXIT_FAILURE;
   }
   char chaine[100];
   while(1){
      scanf("%s", chaine);
      if (strcmp(chaine,"exit")==0)
         return EXIT_SUCCESS;









      printf("%s\n", chaine);
   }
}
```

Upper case vs. Lower case

```
#include <stdio.h>                                  // printf() et scanf()
#include <stdlib.h>                                      // EXIT_SUCCESS
#include <string.h>                                         // strlen()
#include <ctype.h>                                // tolower() et toupper()

int main (int argc, char *argv[]){
   if (argc != 2){                                  // test ultra minimal..
      printf("Usage: %s <upper|lower>\n", argv[0]);
      return EXIT_FAILURE;
   }
   char chaine[100];
   while(1){
      scanf("%s", chaine);
      if (strcmp(chaine,"exit")==0)
         return EXIT_SUCCESS;
      if (strcmp(argv[1],"upper")==0)
         for (int i=0; i<strlen(chaine); i++)
            chaine[i]=toupper(chaine[i]);



      printf("%s\n", chaine);
   }
}
```

Upper case vs. Lower case

```
#include <stdio.h>                                  // printf() et scanf()
#include <stdlib.h>                                        // EXIT_SUCCESS
#include <string.h>                                           // strlen()
#include <ctype.h>                                 // tolower() et toupper()

int main (int argc, char *argv[]){
   if (argc != 2){                                    // test ultra minimal..
      printf("Usage: %s <upper|lower>\n", argv[0]);
      return EXIT_FAILURE;
   }
   char chaine[100];
   while(1){
      scanf("%s", chaine);
      if (strcmp(chaine,"exit")==0)
         return EXIT_SUCCESS;
      if (strcmp(argv[1],"upper")==0)
         for (int i=0; i<strlen(chaine); i++)
            chaine[i]=toupper(chaine[i]);
      else
         for (int i=0; i<strlen(chaine); i++)
            chaine[i]=tolower(chaine[i]);
      printf("%s\n", chaine);
   }
}
```

Réécriture de strcmp

```
int strcmp_bis(const char *s1, const char *s2)
{
  ??
}
```

Réécriture de strcmp

```
int strcmp_bis(const char *s1, const char *s2)
{
  ??
}
```

```
  while ((*s1 && *s2) && (*s1 == *s2)){
    s1++;
    s2++;
  }
  return *s1 - *s2;
```

hum hum...

Réécriture de strcmp

```
int strcmp_bis(const char *s1, const char *s2)
{
  ??
}
```

```
  while ((*s1 && *s2) && (*s1 == *s2)){
    s1++;
    s2++;
  }
  return *s1 - *s2;
```

hum hum...

Encore plus concis ?

Réécriture de strcmp

```
int strcmp_bis(const char *s1, const char *s2)
{
  ??
}
```

```
  while ((*s1 && *s2) && (*s1 == *s2)){
    s1++;
    s2++;
  }
  return *s1 - *s2;
```

hum hum...

Encore plus concis ?

```
  while((*s1 && *s2) && (*s1 == *s2))
      s1++,s2++;
  return *s1 - *s2;
```

Réécriture de strcmp

```
int strcmp_bis(const char *s1, const char *s2)
{
  ??
}
```

```
  while ((*s1 && *s2) && (*s1 == *s2)){
    s1++;
    s2++;
  }
  return *s1 - *s2;
```

hum hum...

Encore plus concis ?

```
  while((*s1 && *s2) && (*s1 == *s2))
      s1++,s2++;
  return *s1 - *s2;
```

Et en une ligne ?

Réécriture de strcmp

```
int strcmp_bis(const char *s1, const char *s2)
{
  ??
}
```

```
  while ((*s1 && *s2) && (*s1 == *s2)){
    s1++;
    s2++;
  }
  return *s1 - *s2;
```

hum hum...

Encore plus concis ?

```
  while((*s1 && *s2) && (*s1 == *s2))
      s1++,s2++;
  return *s1 - *s2;
```

Et en une ligne ?

```
  return strcmp(s1, s2);
```

:D

Structures

→ Permettent de regrouper plusieurs champs dans une même variable
(∼ équiv. classe C++ sans méthodes).

Ex :

```
struct Date{
  int day;       // ou alors uint8_t (defini dans stdint.h)
  int month;     //
  int year;      // uint16_t
};
```

Structures

→ Permettent de regrouper plusieurs champs dans une même variable
(∼ équiv. classe C++ sans méthodes).

Ex :

```
struct Date{
  int day;       // ou alors uint8_t (defini dans stdint.h)
  int month;     //
  int year;      // uint16_t
};
```

Utilisation :

```
struct Date ma_date;
ma_date.day=26;     // utilisation d'un point pour designer le champ
ma_date.month=8;
ma_date.year=1991;
```

Structures

→ Permettent de regrouper plusieurs champs dans une même variable
(∼ équiv. classe C++ sans méthodes).

Ex :

```
struct Date{
  int day;       // ou alors uint8_t (defini dans stdint.h)
  int month;     //
  int year;      // uint16_t
};
```

Utilisation :

```
struct Date ma_date;
ma_date.day=26;     // utilisation d'un point pour designer le champ
ma_date.month=8;
ma_date.year=1991;
```

→ Exercice : écrire une fonction d'affichage pour les dates.

→ Solution 1 :

```
void afficher_date(struct Date date){
  printf("%02d/%02d/%4d\n",
         date.day, date.month, date.year);
}
```

Utilisation :

```
afficher_date(ma_date);
```

→ Solution 1 :

```
void afficher_date(struct Date date){
  printf("%02d/%02d/%4d\n",
         date.day, date.month, date.year);
}
```

Utilisation :

```
afficher_date(ma_date);
```

→ Solution 2 (pointeur) :

```
void afficher_date(struct Date *date){
  printf("%02d/%02d/%4d\n",
         date->day, date->month, date->year);
}
```

Utilisation :

```
afficher_date(&ma_date);
```

Types énumérés (et exercices)

```
enum Gender {MALE, FEMALE};
```

Types énumérés (et exercices)

```
enum Gender {MALE, FEMALE};
struct Person{
   char first_name[30];
   char last_name[30];
   enum Gender gender;
   struct Date birth_date;
};
```

Types énumérés (et exercices)

```
enum Gender {MALE, FEMALE};

struct Person{
   char first_name[30];
   char last_name[30];
   enum Gender gender;
   struct Date birth_date;
};
```

```
struct Person p;

strcpy(p.first_name, "Ada");
strcpy(p.last_name, "Lovelace");
p.gender = FEMALE;
p.birth_date.day = 10;
p.birth_date.month = 12;
p.birth_date.year = 1815;
```

Types énumérés (et exercices)

```
enum Gender {MALE, FEMALE};

struct Person{
   char first_name[30];
   char last_name[30];
   enum Gender gender;
   struct Date birth_date;
};
```

```
struct Person p;

strcpy(p.first_name, "Ada");
strcpy(p.last_name, "Lovelace");
p.gender = FEMALE;
p.birth_date.day = 10;
p.birth_date.month = 12;
p.birth_date.year = 1815;
```

→ Exercice : écrivez une fonction d'affichage pour les personnes.

Types énumérés (et exercices)

```
enum Gender {MALE, FEMALE};
struct Person{
   char first_name[30];
   char last_name[30];
   enum Gender gender;
   struct Date birth_date;
};
```

```
struct Person p;
strcpy(p.first_name, "Ada");
strcpy(p.last_name, "Lovelace");
p.gender = FEMALE;
p.birth_date.day = 10;
p.birth_date.month = 12;
p.birth_date.year = 1815;
```

→ Exercice : écrivez une fonction d'affichage pour les personnes.

→ Exercice : définissez une structure *Couple* qui pointe sur deux personnes.

Types énumérés (et exercices)

```
enum Gender {MALE, FEMALE};
struct Person{
   char first_name[30];
   char last_name[30];
   enum Gender gender;
   struct Date birth_date;
};
```

```
struct Person p;
strcpy(p.first_name, "Ada");
strcpy(p.last_name, "Lovelace");
p.gender = FEMALE;
p.birth_date.day = 10;
p.birth_date.month = 12;
p.birth_date.year = 1815;
```

→ Exercice : écrivez une fonction d'affichage pour les personnes.

→ Exercice : définissez une structure *Couple* qui pointe sur deux personnes.

→ Exercice : écrivez une fonction qui calcule la différence d'âge dans un couple.

Types énumérés (et exercices)

```
enum Gender {MALE, FEMALE};
struct Person{
   char first_name[30];
   char last_name[30];
   enum Gender gender;
   struct Date birth_date;
};
```

```
struct Person p;
strcpy(p.first_name, "Ada");
strcpy(p.last_name, "Lovelace");
p.gender = FEMALE;
p.birth_date.day = 10;
p.birth_date.month = 12;
p.birth_date.year = 1815;
```

→ Exercice : écrivez une fonction d'affichage pour les personnes.

→ Exercice : définissez une structure *Couple* qui pointe sur deux personnes.

→ Exercice : écrivez une fonction qui calcule la différence d'âge dans un couple.

→ Exercice : tentez d'exécuter le code suivant :

```
struct Person *person;

strcpy(person->first_name, "Linus");
```

Que se passe-t-il ? Pourquoi ? Tentez de résoudre ce problème en insérant une nouvelle instruction entre les deux.

Types énumérés (et exercices)

```
enum Gender {MALE, FEMALE};
struct Person{
   char first_name[30];
   char last_name[30];
   enum Gender gender;
   struct Date birth_date;
};
```

```
struct Person p;
strcpy(p.first_name, "Ada");
strcpy(p.last_name, "Lovelace");
p.gender = FEMALE;
p.birth_date.day = 10;
p.birth_date.month = 12;
p.birth_date.year = 1815;
```

→ Exercice : écrivez une fonction d'affichage pour les personnes.

→ Exercice : définissez une structure *Couple* qui pointe sur deux personnes.

→ Exercice : écrivez une fonction qui calcule la différence d'âge dans un couple.

→ Exercice : tentez d'exécuter le code suivant :

```
struct Person *person;
person = malloc(sizeof(struct Person));
strcpy(person->first_name, "Linus");
```

Que se passe-t-il ? Pourquoi ? Tentez de résoudre ce problème en insérant une nouvelle instruction entre les deux.