Partiel de INF104, 23/02/2015

1h30 – Sans documents

Questions sur le langage C (10 points)

Question 1 (1 point) : On suppose qu'on a créé un fichier exécutable appelé `display_cl` à partir du code C fournit ci-dessous.

```
#include <stdio.h>

int main(int argc, char* argv[]){
  printf("value of argc: %d\n", argc);
  int i;
  for(i=0;i<argc;i++)
  {
    printf("value of argv[%d]: %s\n", i, argv[i]);
  }
  return 0;
}
```

On exécute la commande : `./display_cl –c –d 5`

Quel affichage obtient-on ? Justifier votre réponse en précisant les valeurs prises par les paramètres argc et argv.

Réponse :
L'affichage obtenu est le suivant

```
value of argc: 4
value of argv[0]: ./display_cl
value of argv[1]: -c
value of argv[2]: -d
value of argv[3]: 5
```

Lors du lancement d'une application, un élément est ajouté au tableau argv pour chacune des chaines de caractères contenues dans la commande et séparées par des espaces. La valeur de argc correspond à la taille de ce tableau, soit le nombre de chaines de caractères, séparées par un espace, passées sur la ligne de commande. Ainsi, il y a donc 4 chaines de caractères à stocker dans argv : ''./display_c1'', ''-c'', ''-d'', et ''5''. Ceci justifie la valeur de argc : 4 et l'affichage qui correspond exactement aux 4 séquences de caractères qui composent la commande exécutée.

Question 2 (2 points) : En considérant un programme dont le code est fourni ci-dessous, qu'est-ce qui va s'afficher à l'écran lors de son exécution ? Justifiez votre réponse. Pour information, la représentation en hexadécimal de 257 est : 0x0101. Nous rappelons ici qu'en hexadécimal, les chiffres sont 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, et F.

```c
#include <stdio.h>

void abs(int x)
{
  if(x<0)
    x=-x;
}

int main(int argc, char* argv[]){
  int x = 5;
  abs(x);
  printf("value of x: %d\n", x);
  x=-5;
  abs(x);
  printf("value of x: %d\n", x);
  char c = 257;
  printf("value of c: %d\n", c);
  return 0;
}
```

Réponse :

L'affichage de ce programme ne varie pas en fonction des paramètres passés lors de son lancement : aucun usage n'est fait de argv et argc.

On obtient le texte suivant systématiquement :

value of x: 5
value of x: -5
value of c: 1

Question 3 (1.5 point) : Qu'est-ce qu'une fuite mémoire ? Donner la signature de la fonction C qui permet de réaliser une allocation dynamique de mémoire en C.

Réponse :
Définition du cours :
On appelle fuite mémoire (memory leak) le fait que la mémoire allouée n'est pas entièrement libérée (on ne fait pas autant de free que de malloc)

La fonction C permettant d'obtenir de la mémoire de manière dynamique est malloc :

void* malloc (unsigned int) ;

Question 4 (1.5 point) : On s'intéresse au code C suivant :

```
#include <stdio.h>

short glob_var[5] = {257,255,360,365,460};

int main(int argc, char* argv[]){
  char * var1 = (char*) glob_var;
  short * var2 = (short*) var1;

  printf("size of short: %d\n", (char) sizeof(short));
  printf("value of var1, hexa: %p\n",var1);
  printf("value of var2, hexa: %p\n", (char*) var2);

  var1++;

  printf("value of var1, hexa: %p\n", (char*) var1);

  var2++;

  printf("value of var2, hexa: %p\n", (char*) var2);
  printf("value of *var2, hexa: %04x\n", *var2);
  printf("value of *var1, hexa: %04x\n", *var1);
  return 0;
}
```

Indication sur le code : %p permet d'afficher une adresse mémoire sous la forme d'un nombre hexadécimal. Nous rappelons ici qu'en hexadécimal, les chiffres sont 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, et F. %04x permet d'afficher une valeur avec 4 chiffres en base hexadécimal. Par exemple, l'affichage de la valeur 8 avec le format %04x est : 0008.

On vous donne une partie de l'affichage obtenue lors de l'exécution du programme ci-dessus, à vous de le compléter (en remplaçant les ...) par la valeur qui devrait s'afficher.

```
size of short: 2
value of var1, hexa: C528
value of var2, hexa: C528
value of var1, hexa: C529
value of var2, hexa: ...
value of *var2, hexa: ...
value of *var1, hexa: ...
```

Réponse :

size of short: 2
value of var1, hexa: C528
value of var2, hexa: C528
value of var1, hexa: C529
value of var2, hexa: C52A
value of *var2, hexa: 00FF
value of *var1, hexa: 0001

Question 5 (4 points) : Ecrire le code C qui implémente la fonction **strcpy**. Pour information, la fonction **`strcpy`** copie la chaine de caractères `src` dans la chaine de caractère `dst` (y compris le caractère de fin de chaine : '\0').

Il faut donc que vous complétiez le code ci dessous (en remplaçant les ...) :

```
void strcpy(char *dst, const char *src)
{
   ...
}
```

Réponse :

```
void strcpy(char *dst,const char *src){
 int i=0 ;
 while (src[i] !='\0'){
  dst[i]=src[i] ;
  i++ ;
 }
 dst[i]='\0';
}
```

Questions sur la chaîne de production (10 points)

Question 6 (2 points): Expliquez brièvement la différence entre la compilation et l'édition de lien. Quelle option de **gcc** permet de s'arrêter avant l'édition de lien ?

Réponse :
La compilation transforme les instructions du code source en langage machine (assembleur) lors que c'est possible. La compilation analyse et sépare les différents concepts défini dans le programme pour leur trouver un équivalent compréhensible par la machine: déclaration de variable globales et constantes=réservation d'adresses mémoire, déclaration et définition de fonctions = traduction en instructions assembleur…

Toute référence à une fonction déclarée mais non définie dans le code est acceptée et son traitement est différé à la phase d'édition de lien. L'édition de lien se charge d'assembler le contenu du fichier binaire final à partir de différent fichier dit objet, pour créer un exécutable. C'est durant cette phase que chaque variable et fonction déclarée dans le programme doit pouvoir être associée à une définition (soit dans un fichier objet passé en paramètre, soit via une option indiquant que l'adresse du code correspondant sera fournie à l'exécution de l'application)

Pour ne réaliser que la traitement jusqu'à la compilation on peut utiliser l'option « –c »

Considérons le programme C suivant :

Contenu du fichier main.c

```
#define max 100 ;
#define double(x) 2*(x)
#include <stdio.h>

int sup( int n, int m) ;

int main (int argc, char* argv[]){
  printf("%d", sup (double(argc), max) );
  return 0 ;
}

int sup(int n1, int n2)
{
  if(n1>n2)
    return n1;
  return n2;
}
```

Question 7 (1 point): Ecrivez le code résultant de l'exécution du pré-processeur sur le fichier main.c ci-dessus. Attention, on ignorera la traduction du `#include <stdio.h>`.

Réponse :

```
int sup( int n, int m) ;
int main (int argc, char* argv[]){
 printf("%d", sup (2*(argc), 100;) );
 return 0 ;
}

int sup(int n1, int n2)
{
 if(n1>n2)
  return n1;
 return n2;
}
```

Question 8 (1 point): Après exécution du pré-processeur, la compilation va-t-elle réussir ou échouer dans le cas du code ci-dessus? Justifiez votre réponse.

Réponse :

Ce code ne va pas pouvoir passer la phase de compilation à cause de la ligne printf("%d", sup (2*(argc), 100;));

Cette ligne ne respecte pas les règles de syntaxe du langage C. La cause principale de cette erreur réside dans l'ajout de ' ;' dans la définition de la macro de substitution

On souhaite compiler le programme constitué des fichiers sources suivants :

Contenu du fichier main.c	*Contenu du fichier globs.c*

`char state=0;`
`void print_state();`
 	
`int main(int argc, char* argv[]){`
`  print_state();`
`}`
	`#include <stdio.h>`
	
	`char state=4;`
	`void print_state()`
	`{`
	`  printf("State=%d",state);`
	`}`

Question 9 (1.5 point): On va exécuter la commande :

```
gcc –o appli main.c globs.c
```

Cependant, on sait qu'on va avoir une erreur… En quoi consiste cette erreur? A quelle étape de l'exécution de **gcc** va-t-elle se manifester ?

Réponse :
L'erreur concerne la double définition de la variable state et aura lieu au moment de l'édition de lien.

Question 10 (1.5 point): Comment modifier le code précédent pour résoudre ce problème ? Indiquez, pour un fichier donné, la (ou les) ligne(s) de code que vous devez supprimer et le code que vous utilisez en remplacement des lignes de code enlevées.

Réponse :
Il convient de substituer l'une (et une seule) des deux lignes « char state=4 ; » par « extern char state ; »

Question 11 (2 points): on souhaite écrire un makefile qui compile un programme C à partir de trois fichiers C : main.c, graph.c, path_computation.c. Compléter le makefile suivant en remplaçant les pointillés par le code qui convient.

```
##########################################
CC        = gcc
CFLAGS    = …
LD        = gcc
OFILES    = main.o graph.o path_computation.o
##########################################

clean:
         rm *.o prog

all: prog

#production de l'objet main.o (a completer)
main.o: …
         $(CC) $(CFLAGS) …

# production de l'objet graph.o (a completer)
graph.o: …
         $(CC) $(CFLAGS) …

# production de l'objet path_computation.o (a completer)
path_computation.o: …
         $(CC) $(CFLAGS) …


# production de l'executable prog (a completer)
prog: $(OFILES)
         $(LD) … -o prog
```

Réponse :

```
##########################################
CC      = gcc
CFLAGS  = -c –Wall
LD      = gcc
OFILES  = main.o graph.o path_computation.o
##########################################

clean:
        rm *.o prog

all: prog

#production de l'objet main.o (a completer)
main.o: main.c
        $(CC) $(CFLAGS) main.c

# production de l'objet graph.o (a completer)
graph.o: graph.c
        $(CC) $(CFLAGS) graph.c
```

```
# production de l'objet path_computation.o (a completer)
path_computation.o: path_computation.c
	$(CC) $(CFLAGS) path_computation.c



# production de l'executable prog (a completer)
prog: $(OFILES)
	$(LD)  $(OFILES) –o prog
```

Question 12 (1 point): Quel sera l'effet de l'exécution de la commande make ?

Réponse :

Suppression des fichiers .o et de prog si ils existent