Grp **Nom**
❑A
❑B
❑C

Contrôle court n°3

Calculatrice et documents interdits - Durée 1 heure - Répondre sur la feuille

1. QUESTIONS DE COURS : GENERALITES SUR LA COMPILATION

1.1. Quels sont les différents moyens de faire exécuter un programme à un ordinateur ?
Illustrez votre réponse par les différents langages que vous avez rencontré (Assembleur, C, JavaScript, Java, Matlab, Python, Shell Unix…). Précisez les logiciels nécessaires à l'exécution de ces programmes et les avantages ou les inconvénients de ces techniques.

Interprété en ligne ou ligne par ligne (JavaScript, Matlab, Shell),
avec un interpréteur (lié à l'OS). Particulièrement portable, mais lent.
Compilé (Assembleur, C),
avec un compilateur. Non portable mais très rapide.
Semi-compilé vers un byte-code (Java, Python),
avec un compilateur et une machine virtuelle (lié à l'OS). Portable, plus rapide.
Quand on parle de portabilité, c'est hors sources.
Parfois, même les sources ne sont pas portables (bibliothèques spécifiques à l'OS…)

1.2. Donnez les étapes pour passer d'un algorithme à l'exécution d'un programme (pensez aux éventuelles différences selon les langages et système d'exploitation Linux ou DOS) en donnant quelques exemples d'opérations effectuées ainsi que les types de fichiers créés.

Ecriture des sources
(clavier -> .py, .c, .java).
Précompilation
(concaténation des fichiers d'en-tête, substitution de texte) (.i)
Compilation et optimisation
(traduction des instructions haut niveau en instructions assembleur) (.s .asm .pyc .class)
Assemblage
(traduction en langage machine avec des 'trous') (fichiers objets .o ou .obj)
Edition de liens
(remplissage des 'trous') (fichiers à exécuter)
Sur Linux éventuel changement de droit +x (fichiers exécutable)

2. ANALYSE DE TEXTE

Voici un article publié dans PC Expert du mois d'avril.

> **Le règne du BIOS touche à sa fin**
> Déjà en place sur les stations équipées de processeur Itanium, l'Extensible Firmware Interface (EFI), le successeur présumé du Bios, ne devrait pas tarder à débarquer sur les PC de bureau. Microsystème d'exploitation, l'EFI est capable d'apporter des innovations au moment du démarrage du PC. Il supporte l'affichage en 800 x 600 x 32bits et gère de nombreuses options au démarrage (Ethernet, USB…). De plus, la programmation est réalisée en C interprété, ce qui devrait accélérer, sécuriser et simplifier le développement des mises à jour.

Quel(s) avantage(s) et quel(s) inconvénient(s) voyez vous à utiliser du C interprété plutôt que de programmer en assembleur compilé comme c'était le cas avant ? Expliquez très succinctement.

interprété ? (c'est plus du C alors) On perds l'avantage de la rapidité d'exécution
mais on doit gagner en portabilité (pilote usb)
C plus facile d'écriture que l'assembleur, même si en interprété il y a plus clair.
(il existe des puces qui se programment directement en basic ou en Java)
par ailleurs le C est déjà très répandu dans le milieu des programmeur bas niveau

3. ETUDE D'UN PROGRAMME COMPILE

On a récupéré sous forme de fichier objet une fonction dont le code avait été écrit en C.

3.1. Pourquoi ne peut-on pas directement lire le contenu de ce fichier ?

C'est du code objet, du binaire !

3.2. On aimerait utiliser cette fonction dans d'autres fichiers.
De quel autre fichier devrait-on disposer (à part le fichier source) pour y parvenir facilement ?

Un fichier d'en-tête .h permettrait d'expliquer la forme des paramètres reçus et de la valeur retournée.

Que doit-on faire avec ce fichier ? Et avec le fichier objet ?

Il faut l'inclure avec #include "fonction.h"
Pour utiliser le fichier objet il faudrait faire une édition de lien.

3.3. On essaye de reconstituer le prototype de la fonction en désassemblant le fichier objet.
Avant d'en étudier le code ci-dessous, rappelez les règles générales de compilation du C pour :
- le passage de paramètres

Paramètres empilés par l'appelant et pile rabaissée au retour par l'appelant.

- le retour de valeur

Valeur de retour transmise par un registre (eax ici)

3.4. Voici le code récupéré à partir du fichier objet :

```
@1@0:   push    bp
        mov     bp,sp
@1@1:   sub     sp,2
        mov     word ptr [bp-2],2     ;word = 16 bits
@1@2:   jmp     short @1@7

@1@3:   mov     ax,word ptr [bp+4]
        cdq                           ;préparation pour la division entière
        idiv    word ptr [bp-2]       ;division de ax par le paramètre
@1@4:   cmp     dx,0                  ;dx contient le reste
        jne     short @1@6
@1@5:   mov     ax,0
        jmp     short @1@A
@1@6:   inc     word ptr [bp-2]
@1@7:   mov     ax,word ptr [bp-2]
        imul    ax,word ptr [bp-2]    ;multiplication entière
@1@8:   cmp     ax,word ptr [bp+4]
        jl      short @1@3
@1@9:   mov     ax,1
        jmp     short @1@A

@1@A:   mov     sp,bp
        pop     bp
        ret
```

Entre `@1@3` et `@1@A`, qu'y a-t-il dans la pile en :
- BP–2 ?

une variable locale

- BP ?

l'ancienne valeur de BP, avant l'appel

- BP+2 ?

l'adresse de retour de la fonction

- BP+4 ?

le paramètre

En déduire le prototype de la fonction.

```
int fonction( int ) ;          // un int 16 bits
```

Que fait-elle à votre avis ?

Fonction « est premier »
Retourne 0 ou 1 selon que le paramètre est premier ou non