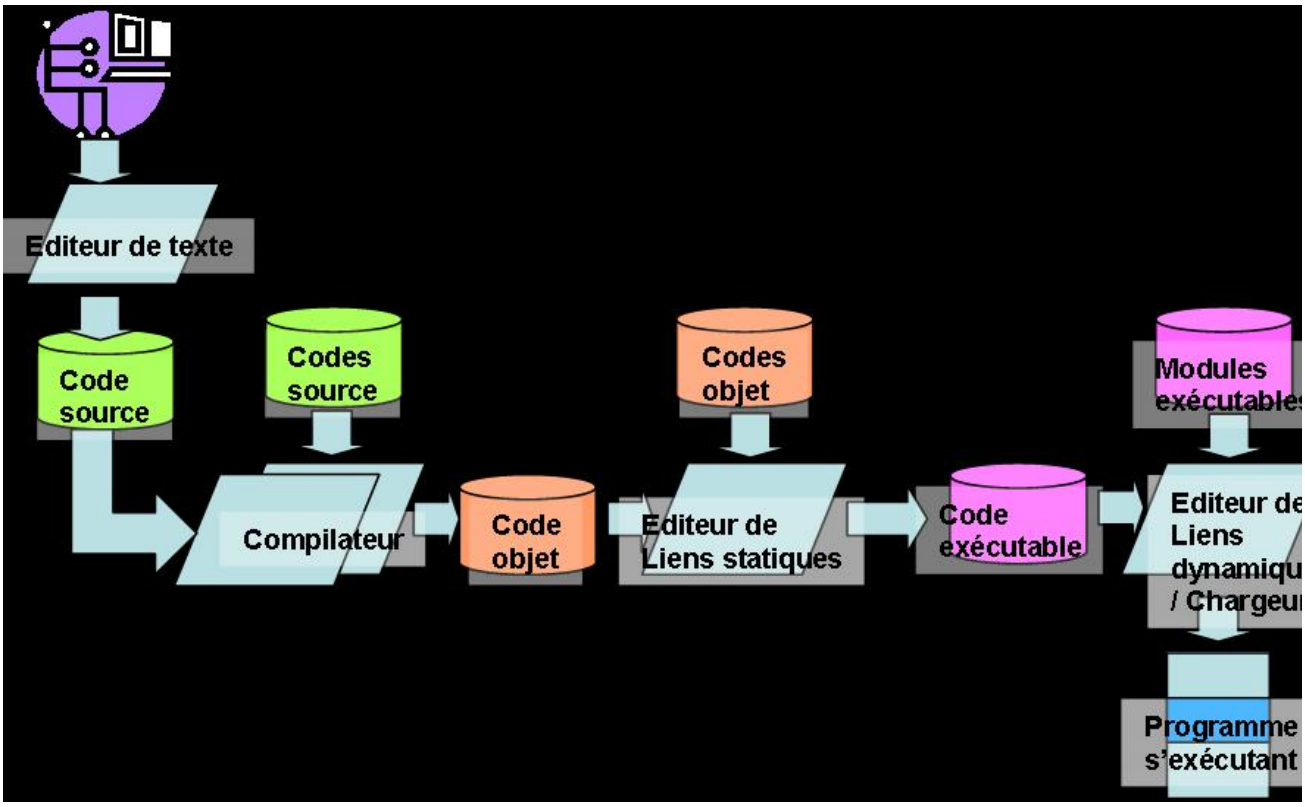
Cours informatique : Maîtrise de Sciences Cognitives 2003-2004

Session 2: "Du texte au programme : la chaîne de fabrication d'un programme C"

Rappel sur l'environnement informatique (sur machine)

1. se connecter avec le login et le mot de passe ;
2. rappel, vous travaillez sur des terminaux, mais vous partagez la même machine pour vos travaux, avec votre propre espace de travail. Utiliser la commande showme
3. lister le contenu du répertoire courant (commande ls)
4. lancer l'éditeur de texte et créer un fichier que vous sauvegardez sous un nom spécifique
5. fabriquer un répertoire spécifique appelé "INFO-SESSION1" (commande mkdir)
6. copier le fichier fabriqué avec l'éditeur de texte dans ce répertoire (commande mv)
7. se positionner dans le répertoire INFO-SESSION1 (commande cd)

Rappel de la chaîne de fabrication d'un programme exécutable (cours)



Les opérations nécessaires pour réaliser un programme jusqu'à l'exécuter sont :

1. Fabriquer le texte source à l'aide d'un éditeur de texte qui permettra de créer un fichier que l'on appellera le code source. Ce code source est un fichier texe. Il revient au programmeur d'écrire son texte en respectant la syntaxe du langage de programmation qu'il a choisi (pour nous donc le langage C). Outre le texte qui sera entré manuellement par lui, le programmeur peut demander l'inclusion d'autres codes sources (des fichiers textes qui auront été fabriqués auparavant et stockés sur le disque) en introduisant des lignes de commande (reconnaissables parce que leur syntaxe est différente) donnant d'une manière ou d'une autre le nom des fichiers textes contenant les codes sources concernés. Dans le cas du langage C, les fichiers sont suffixés par .c : toto.c, mon_premier_code.c sont des noms possibles pour les fichiers qui contiennent les codes source en langage C.
2. Compiler le texte ainsi produit à l'aide d'un programme "compilateur" adapté, c'est-à-dire spécifique au langage de programmation choisi (par exemple pour nous, il s'agira de gcc ou de cc). Le compilateur analyse le texte du fichier source, importe les fichiers sources complémentaires éventuels qui ont été spécifiés et réalise une analyse syntaxique et grammaticale du texte. Si le texte comporte des erreurs de syntaxe ou de grammaire, une liste d'erreurs est produite et il faudra que le programmeur (vous :-) corrige son texte en conséquence avant de faire un nouvel essai. Si la syntaxe et la grammaire sont conformes, alors le texte est traduit dans le langage cible de la machine (le processeur de la machine possède un jeu d'instruction qui lui est propre) produisant ce que nous appelons le code objet dans un fichier reconnaissable en général à son suffixe (par exemple toto.o, mon_premier_code.o dans l'environnement LINUX que nous utiliserons). Le système d'exploitation influe sur la structure du fichier objet car le compilateur doit fabriquer une entête de fichier décrivant les autres codes objet qu'il suppose disponible pour compléter l'objet qu'il vient de fabriquer. Cette entête (son format, la nature des informations, etc.) est dépendante du système d'exploitation. En pratique le compilateur s'exécute en plusieurs passes pour le langage C : une phase de prétraitement (préprocesseur) qui réalise un traitement des lignes "spéciales" (lignes commençant par #, lignes de commentaires typiquement) ; un traducteur en langage assembleur (instructions machines liées au processeur disponible) ; un optimiseur de code (qui remodèle la traduction en fonction d'un modèle de programmation de

l'unité centrale de l'ordinateur).

3. Lier le code objet fabriqué avec les autres codes objets implicitement nécessaires. Le programmeur fait en effet appel à des fonctions prédéfinies (qu'il n'a pas écrite lui-même dans le code source en cours) qui ont été écrites par d'autres, compilées et disponibles comme codes objets "préfabriqués" (notion de bibliothèque de modules objets). C'est par exemple systématiquement le cas pour les fonctions d'entrée-sortie (écrire des choses à l 'écran, saisir des choses au clavier). Les références directes ou indirectes aux modules de la bibliothèque s'appellent des "liens". Le programme s'appelle "éditeur de liens" (linker) pour cette raison. On dit aussi qu'il résoud les liens, c'est-à-dire qu'il cherche à trouver les modules objets correspondants aux liens laissés en suspens dans le code. Si des liens ne sont pas résolus (les modules objets requis ne sont pas disponibles), alors une liste d'erreurs est produite et le programmeur devra soit reprendre son source pour corriger (le plus souvent indirectement) son texte qui a généré la demande du module objet concerné, soit donner explicitement au programme d'édtion de lien, le chemin exact où il pourra trouver le module concerné (le chemin signifie le nom du fichier concerné avec la succession des répertoires/sous-répertoires qui y conduisent). Si tous les liens sont résolus, alors l'éditeur de lien produit un "code exécutable" dans un fichier avec parfois un suffixe explicite (par exemple dans les systèmes microsoft .exe). Dans notre environnement LINUX, ces fichiers n'ont pas de suffixe obligatoire. Si le nom du fichier d'exécutable n'a pas été spécifié par le programmeur un nom par défaut est donné (c'est "a.out" dans LINUX). Ce fichier est fabriqué avec une entête qui précise quels seront les autres modules exécutables qu'il faudra mettre en relation quand il s'agira de lancer effectivement l'exécution du programme. Cet entête est totalement lié au système d'exploitation (en effet, le lancement du programme déclenchera la création d'un "processus" qui représentera le programme "en train" de s'exécuter).
4. Lancer le programme. Le lanceur de programme est un programme (!) qui assemble les modules exécutables nécessaires à l'exécution du code exécutable lancé (en effet, la plupart des programmes s'exécutant partageant des fonctions d'entrée-sortie par exemple, c'est le même module exécutable qui pilote l'entrée-sortie qui sera lié "dynamiquement" au moment même du lancement au code exécutable). Si un module exécutable n'est pas disponible, il y aura une erreur. Si tous les modules exécutables nécessaires sont disponibles, alors le lanceur charge en mémoire centrale les instructions correspondantes et alloue la place mémoire pour les données telles qu'elles ont été décrites par le programmeur. Dynamiquement, il est possible pendant l'exécution de demander l'ajout de nouveaux modules ou l'allocation de place mémoire supplémentaire en fonction du déroulement du programme.

Exercice 1 : mon_premier_programme (sur machine)

Code Source

```
#include <stdio.h>

/* cette ligne de commande indique qu'il faudra inclure le code sou
dans le fichier source stdio.h */
/*le couple de caractères /et *est utilisé pour signaler le début d'une
commentaires */
/*le couple *et /est utilisé pour signaler la fin d'une ligne de caractè

/* déclaration de la fonction "principale" du code, c'est-à-dire la pre
partie du code qui sera lancée */
```

```
void main()

{
/*les accolades permettent de structurer en "blocs" les différentes pa
code qui constituent
un ensemble : on pourrait traduire l'accolade ouvrante par "begin" e
fermante par "end" */

printf("Salut tout le monde \n");

/*printf est une fonction prédéfinie qui nécessite la présence de sa d
ce qui est le rôle du fichier stdio.h */

return 0;

}
```

Exercice 2 : mon_deuxième_programme (sur machine)

Ce qu'il faut savoir :

- stdin = flux d'entrée standard (en général le clavier)

- stdout = flux de sortie standard (en général l'écran)

- stderr = flux des messages d'erreurs standard (en général l'écran également)

Le code :

```
#include <stdio.h>

int main()

{

char caractere;

/* il est interdit de mettre des accents aux noms des entités que
vous déclarez*/

printf("merci de taper un caractère au clavier :/n");

caractere = getc(stdin); /* la fonction getc permet de lire UN
```

caractère sur un flux d'entrée*/

printf("le caractere saisi est : %c /n", caractere);

/ le signe % est interprété par la fonction printf comme signalant un format d'impression particulier/

/dans ce programme, %c signale qu'il s'agit d'afficher un symbole alphanumérique (un caractère)/

return 0;

}

Exercice 3 Notion de boucle

#include <stdio.h>

int main()

{

char caractere;

char bidon; /* cette variable va servir à "consommer" le caractère 'return' qui est tapé pour*/

/*la prise en compte de fin de frappe*/

int i; /* int permet de déclarer une variable de type entier */

i=0; /* initialisation du compteur de boucle*/

while (i<5) /* début de la boucle while*/

{ printf("taper un caractère : \n");

caractere = getc(stdin); /*récupération du caractère tapé*/

bidon=getc(stdin); /* consommation du return !*/

printf("le caractere saisi est : %c /n", caractere);

```
i=i+1;
}/* fin de la boucle while*/
return 0;
}
```

Exercice 3 : La boucle FOR

```
#include<stdio.h>
int main()
{
char c;
printf("Entrez un caractère : \n (p pour quitter) \n");
for (c=' ';c!='p'; ) /* Pour (c étant initialement avec la valeur '
'(espace);jusqu'à ce que c soit différent de la valeur 'p'; pas
d'opération particulière à chaque itération*/
{
c=getc(stdin);
putc(c,stdout);
}
printf(" /n au revoir /n");
return 0;
}
```