FONCTIONS ET PROCEDURES

Appel: Il faut

- calculer les paramètres
- aller au début de la procédure
- après avoir stocké l'adresse à laquelle il faut rentrer à la fin!

En général il existe une instruction pour faire les 2 derniers; L'instruction `GO $X, $Y, $Z` fait les deux dernières opérations en une: elle saute vers l'adresse `Y+Z` après avoir stocké l'adresse de l'instruction qui suit le `GO` en `X`.

Pour rentrer il suffirait de faire `GO $X, $X, 0` qui saute à l'adresse stocké en `$X` (avec effet de bord de changer ce registre mais après avoir récupéré son contenu!) Mais cela ne suffit pas si cette procédure en a appelé une autre; Il faut stocker cette adresse dans une location différente pour chaque appel (même plusieurs appels récursifs de la même procédure); on a donc une chaîne d'adresses qui permet de rentrer d'une chaîne d'appels.

Il faut, donc, avoir une méthode de charger l'adresse de la procédure dans un registre. L'instruction `GETA $X, proc` où `proc` est l'étiquette de la procédure convient.

Résultat: Il faut qu'une fonction laisse son résultat dans une location connue; le code qui avait appelé la fonction doit récupérer ce résultat (comme la valeur d'une expression); (avant l'exécution d'autre code qui pourrait utiliser la même location).

LE PROBLEME DES PARAMETRES

Nécessité de code efficace pour accéder aux valeurs ou les changer

La pile est essentielle dans tout langage récursif
Les 4 types d'information sur la pile: variables, paramètres, sous-expressions et données de contrôle
4 piles ??? facile à comprendre mais peu souhaitable

Initialisation et accès: dans le cas simple (comme C), il suffit d'avoir un pointeur à la fin de la pile (à la dernière bloc de la pile), de préférence stocké dans un registre: l'accès à une variable locale est par `LD $T, $pile, I`;
(un bloc de la pile pour une fonction, même s'il y a des variables locales à des {...} imbriqués)
avant le **GO**, les paramètres sont calculés et

leurs valeurs stockées dans les adresses après `$pile` $+$*espace pour les variables locales* et `$pile` est incrémenté; dans la procédure le code pour accéder à un paramètre (le $i$ième) pourrait être `LD $T, $pile, -I`;
(si le nombre et les types des paramètres étaient connus, les paramètres auraient pu être traités comme les variables locales)

à la fin de l'appel `$pile` est décrementé (de combien et par quel code?)

si la fonction appelée le fait, il faut avoir empilé la taille des paramètres
(ou un pointeur au bloc précédent de la pile: la chaîne dynamique)
sinon difficile de parcourir la pile, par ex. pour déboguer.

Au début de la procédure la valeur en `$X` est aussi transféré à la pile; il faudra la récupérer avant l'instruction `GO` à la fin.

Paramètres ref: on stocke l'adresse au lieu de la valeur et le code de la procédure ajoute un niveau d'indirection.

ACCES A DES VARIABLES NI LOCALES NI STATIQUES

Les langages qui le permettent: algol60 et ses descendants (Pascal, modula-2,... Ada, ...): l'imbrication de blocs ne donne pas de problème; c'est l'imbrication de déclarations de fonctions.

Référence par niveau textuel: le compilateur (les attributs) peut calculer, pour chaque variable, son *niveau lexical* et son *décalage*; ces deux suffisent à l'exécution pour déterminer la variable; il faut trouver le moyen d'accéder à cette variable

L'adresseur: un tableau avec un élément pour chaque niveau lexical utilisé dans le programme (c'est-à-dire le niveau maximum d'imbrication de déclarations de fonctions); cet élément est un pointeur à la section de la pile active pour une fonction de ce niveau: donc le code pour accéder à une variable:

```
LD $T, $adr, I
LD $T, $T, J
```

(Mais on évite de répéter les `LD $T, $adr, I` pour les niveaux `I` utilisés souvent, en choisissant un registre différent)

TROUVER LA BONNE ACTIVATION

Récursion: Il peut exister plusieurs activations de procédures du même niveau lexical; de la même procédure (récursion normale) ou différentes procédures (récursion mutuelle);
presque toujours c'est l'activation la plus récente qu'il faut utiliser

Pas toujours l'activation la plus récente!
(une fonction peut être un paramètre): après avoir été passée entre plusieurs procédures, elle est enfin appelée; quelles sont les variables accessibles pendant l'appel? réponse normale: les variables qui étaient accessibles là où le paramètre a été passé (la première fois).

La chaîne statique: chaque invocation dans la pile contient un pointeur à l'invocation au niveau lexical inférieur (par 1) dont il dépend; ce pointeur doit être partie de l'information passée pour un paramètre qui est une procédure.

A noter que dans algol-60, la méthode par défaut de transmission de paramètres équivalait à transmettre une fonction qui était exécutée chaque fois qu'il fallait avoir la valeur du paramètre (!)

RECONSTRUCTION DE L'ADRESSEUR

Suivre la chaîne statique pour trouver la bonne activation à chaque niveau lexical.

Appel d'une procédure: pour construire l'adresseur.

Retour d'une procédure: pour réparer l'adresseur éventuellement modifié par la procédure appelée.

A la fin normale d'un appel, la fonction appelée peut réparer elle même la modification qu'elle avait effectuée à l'adresseur; mais ça ne suffit pas quand il y a des `goto` avec destination non-locale.

Principe à appliquer (autant que possible): les programmes qui n'utilisent pas des facilités coûteuses ne les paient pas.