Annexes NASM

I. Données

I.A. Constantes

```
NB_VARS  = nombre de variables du programme
NB_SYMDESC = Nombre de descripteurs de symboles du programme
SYM_OFFSET = constante a ajouter a l'adresse du descripteur de symbole
pour récupérer l'indice de début de chaîne dans la zone mémoire
SYM_LENGTH = constante a ajouter a l'adresse du descripteur de symbole
pour récupérer la longueur de la chaîne
NB_NODES = nombre de noeud que l'on peut créer
NODE_USE = constante a ajouter a l'adresse du descripteur de noeud pour
récupérer le nombre de pointeurs sur le noeud
NODE_HEAD = constante a ajouter a l'adresse du descripteur de noeud pour
récupérer l'adresse du sous-arbre gauche
NODE_TAIL = constante a ajouter a l'adresse du descripteur de noeud pour
récupérer l'adresse du sous arbre droit
NB_INPUTS = nombre de paramètres du programme et donc de la fonction
main
NB_OUTPUTS = Nombre de résultats du programme,
```

I.B. Variables

```
debug = Booléen conditionnant l’affichage des statistiques
malloc = nombre de demande d’espace mémoire pour un nouveau noeud
ralloc = nombre de fois ou une allocation
list_size = nombre de noeud dans la freelist
```

I.C. Pointeurs

```
sym_free = pointeur sur la zone libre contenant les chaînes de
caractères
node_free = pointeur vers le début de la zone libre contenant les noeud
node_freelist = pointeur vers le premier élément de la freelist
desc_free = pointeur vers la zone libre des descripteurs de symboles
```

I.D. Registres spécialisés

```
R15 = Registre contenant l'adresse du symbole NIL
R14 = Registre contenant l'adresse d'un noeud (cons NIL NIL) valant True
R8 = Adresse vers la première variable de la fonction courante
R9 = Adresse de la fin des variables courantes
R10 = Offset du prochain argument à lire dans le buffer de passage de
paramètres
R11 = Offset du prochain argument à écrire dans le buffer de passage de
paramètres
```

II. Structures de données

II.A. Descripteur de noeud

Compteur de références	Adresse de HEAD	Adresse de TAIL
8 octets	8 octets	8 octets

II.B. Descripteur de symboles

Offset dans la zone mémoire	Longueur de la chaîne
8 octets	8 octets

II.C. Tableau de variables

Pointeur vers une zone mémoire réservé aux variables. Suite de Quad-word(8 octets) qui sont les adresse du contenu des variables. Ces adresses peuvent être des adresses de noeud ou des adresses de symboles

II.D. Tableau de descripteurs de symboles

Suite de paire de descripteurs de symboles

II.E. Buffer

Tableau d'adresse utilisé pour le passage de paramètres à une fonction, l'affectation multiple et même au lancement. A l'aide des registre R10 et R11, ce buffer fonctionne comme une file. L'écriture se fait avec la fonction write et la lecture avec read.

III. Gestion mémoire

III.A. Désallocation / allocation mémoire

Lors d'une affectation, on décrémente le compteur de référence puis si on atteint 0 on ajoute le noeud à la freelist.

Lors d'une construction d'arbre (cons), la première chose est de tester l'égalité entre node_free et node_freelist. S'ils sont égaux la freelist est donc vide on ajoute donc le noeud au début de la zone libre, sinon on supprime le noeud en tête de freelist, puis décrémente le nombre d'utilisation de ses fils et on les ajoute à la freelist si leur compteur tombe à 0 et enfin on remplace ce noeud par le nouveau.

Légende : gras = Instructions WHILE

node_free = N3

node_freelist = [N3]

V1 = (cons (cons nil nil) (cons nil nil))

V1 = N2

N0			N1			N2			N3			N4		
1	nil	nil	1	nil	nil	1	N0	N1						

V1 = nil

node_freelist = [N2,N3]

N0			N1			N2			N3			N4		
1	nil	nil	1	nil	nil	N3	N0	N1						

V1 = (cons nil nil nil nil)

node_freelist = [N3]

N0			N1			N2			N3			N4		
1	nil	nil	1	nil	nil	0	nil	nil						

node_freelist = [N0, N1, N3]

N0			N1			N2			N3			N4		
N1	nil	nil	N3	nil	nil	0	nil	nil						

node_freelist = [N1, N3]

N0			N1			N2			N3			N4		
0	nil	N2	N3	nil	nil	1	nil	nil						

node_freelist = [N3]

N0			N1			N2			N3			N4		
1	nil	N2	0	nil	N0	0	nil	nil						

V1 = N1

N0			N1			N2			N3			N4		
1	nil	N2	1	nil	N0	0	nil	nil						

IV. Librairie BinTree

IV.A. Légende

fonction	PARAM1	PARAM2	RETURN
	Registre	Registre	Registre
description de la fonction			

IV.B. Gestion mémoire

cleanFun	Void	Void	Void
	Void	Void	Void
Remet les cases mémoires utilisée par la fonction à nil			

chgVar	Index	Void	Void
	RSI	Void	Void
Appelée au changement de valeur de la {Index}-ième variable du contexte courant. Cette fonction décrémente son compteur de référence et si il tombe à 0 gère son ajout dans la free list.			

decNodeUse	Node	Void	Integer
	RBX	Void	RAX
Décrémente le compteur de référence du noeud {Node} et retourne la valeur du compteur après la décrémentation.			

delMem	Node	Void	Address
	RBX	Void	RCX
Supprime {Node} de la freelist et y ajoute ses head et tail si leur compteur de référence tombe à 0.			

IV.C. Procédures du code WHILE

cons	Head	Tail	Address
	RSI	RDI	RBX
Construit un nouveau noeud avec {Head} et {Tail} et retourne son adresse			

head	Node	Void	Head
	RSI	Void	RBX
Retourne le {Head} du noeud ou nil si {Node} est un symbole.			

tail	Node	Void	Tail
	RSI	Void	RBX
Retourne le {Tail} du noeud ou nil si {Node} est un symbole.			

read	Index	Void	Void
	RSI	Void	Void
Range dans la {Index}-ième variable du contexte la valeur en tête du fun_buffer			

write	Index	Void	Void
	RSI	Void	Void
Ajoute la {Index}-ième variable du contexte en fin du fun_buffer			

equals	Node1	Node2	Boolean
	RSI	RDI	RAX
Effectue une comparaison en profondeur des arbres dont les racines sont {Node1} et {Node2}. Renvoi 1 si les arbres sont égaux 0 sinon.			

V. Librairie Launcher

parseArgs	Void	Void	Void
	Void	Void	Void
Fonction lancée après l'initialisation des données, elle s'occupe de récupérer les paramètre et de les convertir en arbres binaires en fonction du type de paramètre et les ajoutes au fun_buffer pour que la fonction main puisse les lire.			

parseCons	Address	Length	Address
	RSI	RDI	RBX
Si un paramètre est de type "(cons a b c)" cette fonction est appelé avec {Address} pointant vers le début de la chaîne de caractère, {Length} la longueur de la chaîne et retourne l'{Address} de l'arbre représenté par cette expression.			

asciiToInt	Address	Length	Value
	RSI	RDI	RAX
Si un paramètre est de type "876" cette fonction est appelé avec {Address} pointant vers le début de la chaîne de caractère, {Length} la longueur de la chaîne et retourne la valeur {Value} du nombre représenté.			

intToAscii	Value	Void	Length
	EAX	Void	RDX
Converti la valeur dans EAX en sa représentation ASCII à l'adresse argv et retourne la longueur de cette représentation			

intToBin	Value	Void	Address
	RAX	Void	RBX
Converti la valeur dans RAX en sa représentation en arbre binaire à et retourne l'adresse de cet arbre			

binToInt	Node	Void	Value
	RSI	Void	RAX
Retourne la valeur numérique représenté par l'arbre ayant {Node} pour racine			

printBinTree	Node	Tab	Void
	RSI	RAX	Void
Affiche l'arbre de racine {Node} avec une tabulation générale de {Tab}(Récursivité)			

printBinString	Node	Void	Void
	RSI	Void	Void
Affiche la représentation (concaténation des représentation des head et tail) sous forme de chaîne de caractère de l'arbre {Node}			

addSym	Address	Length	Address
	RSI	RDI	RBX
Ajoute un nouveau symbole au programme et retourne l'adresse du descripteur de ce symbole.			