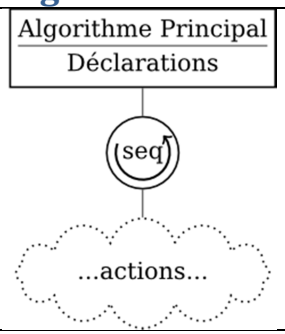
Résumé : Algorithmique (pseudocode, arbre programmatique) → Langage C

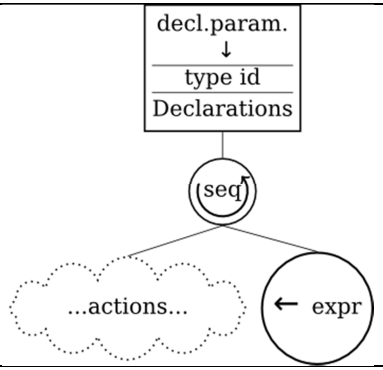
Algorithme principal / fonction principal d'un programme C

Algorithme Principal *Déclarations* *Début* *…actions…* *Fin*		`// directives` **`int main ()`** **`{`** `  // déclarations` `  // instructions` `  return 0 ;` **`}`**

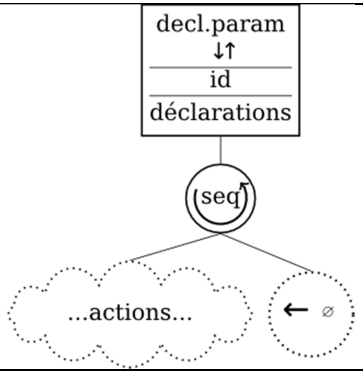
Identifiants (id)

- Commencent par une lettre, comporte ensuite éventuellement des lettres, des chiffres ou « _ »

Fonction / fonction d'un programme C

Fonction id (décl. param.) : type *Déclarations* *Début* *…actions…* *retourner expr* *Fin*		`type id (décl.param.)` **`{`** `  // déclarations` `  // instructions` `  return expr ;` **`}`**

Procédure / fonction sans retour d'un programme C

Procédure id (décl. param.) *Déclarations* *Début* *…actions…* *[retourner] // quitter* *Fin*		**`void`** `id (décl. param.)` **`{`** `  // déclarations` `  // instructions` *`[return;] // quitter`* **`}`**

Résumé : Algorithmique (pseudocode, arbre programmatique) → Langage C

Types de données natifs (type)

entier	**[signed / unsigned] char** **[signed / unsigned] short** **[signed / unsigned] int** **[signed / unsigned] long**
réel	**float, double**
caractère	**char**

Autres types :

chaine(20)	**char**[20] *(cf. tableaux)*
booléen	**bool** *(nécessite stdbool.h)*

Valeurs littérales : exemples

entier	-100, 2013 **0x**hexa, **0**octal
réel	3.14, 3f, 1.5E-12
caractère	'c', '\n', '\t'
chaine	"bonjour à tous !"
booléen (vrai, faux)	**true**, **false**

Déclarations de constantes et variables

const[ante] ID : type ← expr	**#define** ID (expr) **const** type ID = expr;
var[iable] id : type	type id ;

Affectation

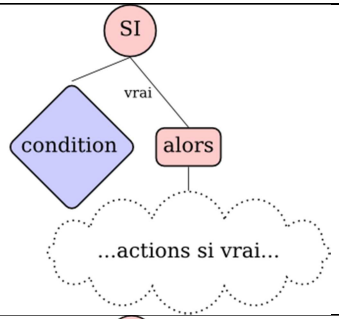
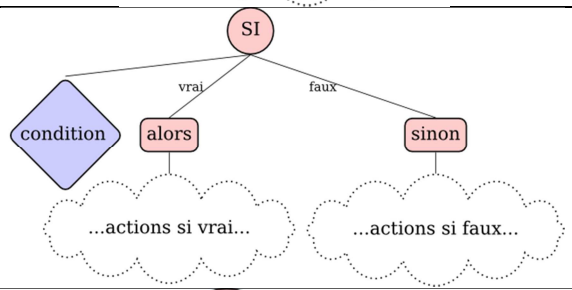
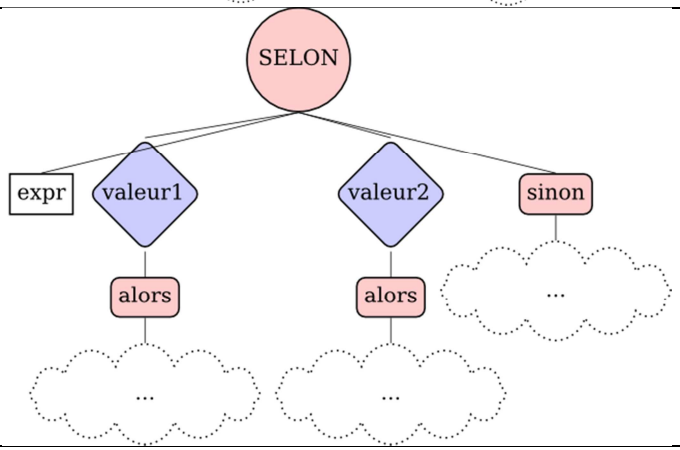
id_var ← expr	id_var ← expr	id_var = expr ;

Expressions et opérateurs (expr)

numérique	*+, -, *, /, %*	**+, -, *, /, %**
logique / condition	*=, !=, <, <=, >, >=*	**==, !=, <, <=, >, >=**
connecteurs logique	*et, ou, non*	**&&, ||, !**

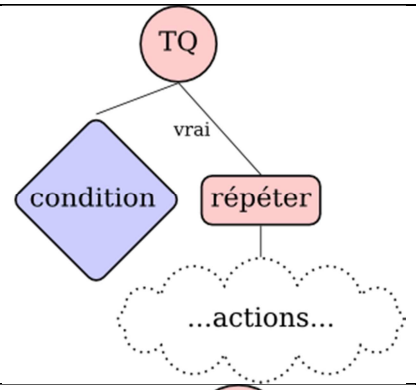
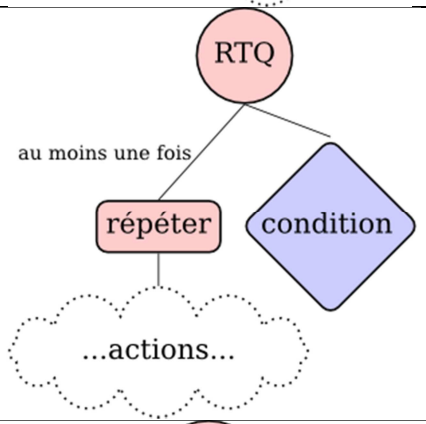
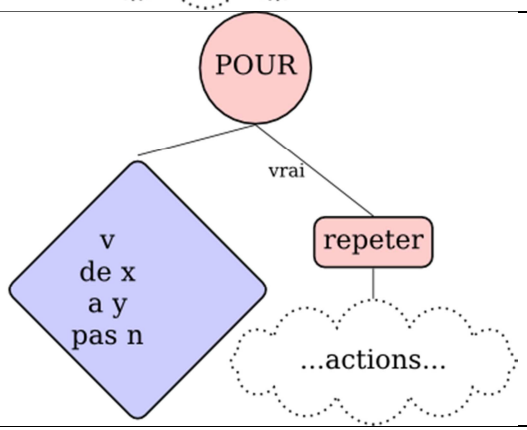
Résumé : Algorithmique (pseudocode, arbre programmatique) → Langage C

Structures de contrôle conditionnelles

si condition *Alors…* *finSi*		`if (condition)` `{ … ; }`
si condition *Alors…* *Sinon…* *finSi*		`if (condition)` `{ … ; }` `else` `{ … ; }`
Selon expr *cas valeur1 : …* *cas valeurN : …* *cas sinon : …* *finSelon*		`switch (expr_numérique)` `{` `case valeur1 : … ; [break ;]` `case valeurN : … ; break ;` `default : …; break;` `}`

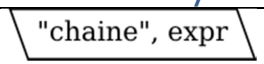
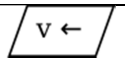
Résumé : Algorithmique (pseudocode, arbre programmatique) → Langage C

Structures de contrôle itératives

tantque condition *…actions…* *finTantque*		`while (expr_logique)` `{` `  …instructions… ;` `}`
repeter *…actions…* *tantque condition*		`do` `{` `  …instructions… ;` `} while (expr_logique);`
pour v de x a y pas de n *finPour*		`for(v=x ;v<=y ;v=v+n)` `{` `  …;` `}`

Résumé : Algorithmique (pseudocode, arbre programmatique) → Langage C

Autres fonctions des bibliothèques : entrées/sorties, nombres aléatoires

ecrire « chaine », expr *afficher « chaine », expr*		`printf("…%format…",expr) ; // stdio`
		`puts("chaine") ; // stdio`
lire v *saisir v*		`scanf("%format", &v) ; // stdio`
lire c *saisir c*		`c = getchar() ;` `while(getchar() != '\n'); // vider le buffer`
	entier *reel* *caractère* *chaine*	`%i, %d , %ld, %u, %lu (autre exemple : %6d)` `%f, %lf, %e, %E (autre exemple : %3.2f)` `%c` `%s`
alea(1,10)		`srand((unsigned) time(NULL)); // stdlib et time` `nb = rand()%10 + 1; // stdlib`

Structures de données complexes (1) – tableau

Tableau de N éléments

var[iable] type id [N]	`type id[N] ;`

Tableau de NxM éléments

var[iable] type id : [N] [M]	`type id[N][M] ;`

Structures de données complexes (2) – structures

TYPE id *déclaration des variables* *finTYPE*	`typedef` `struct id {` `  déclaration des variables ;` `} id ;`

Résumé : Algorithmique (pseudocode, arbre programmatique) → Langage C

Fichiers texte

Déclaration

var[iable] id : FICHIER	**FILE *** id; *(id est un pointeur sur fichier)*

Ouvrir

id ← OUVRIR « nom » EN mode *où « mode » parmi :* *LECTURE* *ECRITURE* *AJOUT*	id = **fopen**("nom", mode) ; *où « mode » parmi :* **"r"** *pour* read **"w"** *pour* write **"a"** *pour* append

Ecrire

ECRIRE_FICHIER id, var1, var2	*écriture d'une chaine unique :* **fputs**(var1,id) ; *écriture formatée :* **fprintf**(id,format, var1, var2) *où "format" est identique au format de printf*

Lire

LIRE_FICHIER id, var1, var2	*lecture d'une chaine unique :* **fgets**(ch, longueur, id) ; *où « ch » est une chaîne (tableau de caractères) et « longueur » la longueur récupérée)* *utilisation conjointe de sscanf pour extraire les éléments de la chaine :* **sscanf(**ch, format,&var,&var2**)** *lecture formatée :* **fscanf**(id, format, &var1, &var2) *lecture octet par octet :(char c)*

	`c = fgetc(id)`

Fermer

FERMER id	`fclose(id) ;`

Gérer les erreurs de base

FIN_DE_FICHIER id	`feof(id)`
après l'ouverture	`if (id==NULL)` `{` `  perror("err.ouverture") ;` `  return 9999;` `}` *où perror envoie le message sur stderr (au lieu de stdout)*

Fichiers binaire (instructions spécifiques)

Ouvrir

id ← OUVRIR « nom » EN mode *où « mode » parmi :* *LECTURE BINAIRE* *ECRITURE BINAIRE* *AJOUT BINAIRE*	`id = fopen("nom", mode) ;` *où « mode » parmi :* `"rb"` *pour* `read binary` `"wb"` *pour* `write binary` `"ab"` *pour* `append binary`

Ecrire

ECRIRE_FICHIER id, enreg	*écriture d'un enregistrement :* `fwrite(&enreg,sizeof(typeEnreg), 1, id) ;` *où "enreg" est la variable de type structure* *typeEnreg est le type de donnée structure* *1 le nombre d'enregistrements écrits (un*

	seul ici) *id est le fichier ouvert*

Lire

LIRE_FICHIER id, enreg	*lecture d'un enregistrement :* `fread(&enreg, sizeof(typeEnreg),1, id) ;` *où "enreg" est la variable de type structure* *typeEnreg est le type de donnée structure* *1 le nombre d'enregistrements lus (un seul ici)* *id est le fichier ouvert*

Se positionner à l'enregistrement n

POSITIONNER_FICHIER id, noEnreg	*se positionner à un certain enregistrement (indispensable en cas de modification : après lecture et avant écriture):* `fseek(id, noEnreg, SEEK_SET) ;` *où id est le fichier ouvert* *"noEnreg" est l'offset (déplacement) en octets par rapport au point de démarrage* *SEEK_SET : constante indiquant que l'offset démarre au début du fichier*

Donner la position du curseur dans le fichier

POSITION id	`ftell(id) ;`

Résumé : Algorithmique (pseudocode, arbre programmatique) → Langage C

Fonction

Fonction id (décl. param.) : type *Déclarations* *Début* *retourner expr* *Fin*	`type id (décl.param.)` `{` `  // déclarations` `  // traitement` `  return expr ;` `}`

Procédure

Procédure id (décl. param.) *Déclarations* *Début* *Fin*	`void id (décl. param.)` `{` `  // déclarations` `  // traitement` `  [return;] // quitte` `}`

Définition des sous-programmes :

- *decl.param*= déclaration des paramètres formels (définition des valeurs attendues par le sous-programme)

Appel d'un sous-programme

id (param.réels)

- param.réels (ou paramètres effectifs) = valeurs réelles passées et distribuées aux paramètres formels sous forme de variables, valeurs littérales, expressions.

Modes de passage des paramètres (decl_param)

(ENTREE age : entier) *la valeur du paramètre age sera simplement utilisée (lue)*	*mode se passage par valeur ou par référence (pointeur) constante :* `(int age)` `(const int * ptrAge)`
(SORTIE age : entier) *la valeur du paramètre age sera fournie (écrite) par le sous-programme*	*mode de passage par référence (pointeur)* `(int * ptrAge)`

Résumé : Algorithmique (pseudocode, arbre programmatique) → Langage C

(ENTREE/SORTIE age : entier) *la valeur du paramètre age sera utilisée et modifiée par le sous-programme*	

Résumé : Algorithmique (pseudocode, arbre programmatique) → Langage C

Pointeurs : déclarations

VAR idPtr : POINTEUR VERS type	`type * idPtr = NULL ;` *Remarque : lorsqu'on passe un tableau en paramètre, on passe en fait, l'adresse de son premier élément.*

Pointeurs : initialisation

idPtr ← ADRESSE DE idVar	`idPtr = & idVar;` *& est l'opérateur de récupération de l'adresse mémoire d'une variable (déjà utilisé dans scanf)* *Les opérateurs arithmétiques d'addition et de soustractions sont applicables aux pointeurs*

Pointeurs : déréférencement

DEREFERENCER idPtr ← valeur	`*idPtr` = *valeur ;*

Pointeurs : allocation dynamique

idPtr ← ALLOUER nb type	`idPtr = (type *) malloc(nb * sizeof(type));`

Pointeurs : desallocation

DESALLOUER idPtr	`free(idPtr);` `idPtr = NULL;`

Pointeurs : sélection des membres d'un type structure

idPtr->idMembre = valeur	`(*idPtr).idMembre = valeur;` *ou* `idPtr->idMembre = valeur;`