Université du Québec (UQ)
École de technologie supérieure
Service des enseignements généraux
Local B-2500 - 396-8938
Site Internet : http://www.seg.etsmtl.ca/inf115/

INF145 Programmation avancée et langage C

MOTS RÉSERVÉS ET INSTRUCTIONS AU PRÉPROCESSEUR

1. DESCRIPTION DU DOCUMENT

Ce document énumère l'ensemble des mots réservés du langage C (ANSI 89) et du préprocesseur. Pour chacun d'eux, il décrit brièvement son intérêt et donne une référence bibliographique. Les mots en gras doivent être connus dès le début de la session.

Les références proviennent du livre :

DRIX, Philippe (1993) **Le Langage C ANSI – Vers une pensée objet en Java**, 3e édition, InterEditions.

Pour plus d'informations sur les commandes du préprocesseur, le livre suivant est recommandé :

HARBISON III, Samuel P. & STEELE JR., Guy L (2002) **C – A reference manual**, 5ème édition, Prentice Hall

2. MOTS RÉSERVÉS DU LANGAGE C

Les mots en police grasse doivent être connus dès le début de la session.

auto : Classe d'allocation par défaut des variables locales (chapitres IX–3, IX–5.a et et IX–8.b).

break : Provoque la fin des instructions *switch*, *while*, *do* et *for* (chapitre V–7).
Voici un exemple de code qui affiche les valeurs de 1 à 10 en utilisant un *break* :

```
int i = 1;

while (1)         /* Boucle infinie. */
{
      printf("%i\n", i);

      ++i;
      if (i > 10)
            break;
}
```

On préférera le code suivant qui évite l'utilisation du *break* :

```
int i;

for (i = 1; i <= 10; ++i)
      printf("%i\n", i);
```

Voir le mot *switch*.

case : Sert à énumérer les différents choix d'un *switch* (voir le mot *switch*).

char : Type de donnée occupant au minimum 1 octet et permettant de représenter des valeurs entières allant de 0 à 255 ou de -128 à 127 (chapitres II–1.c et II–2).

const : Qualificatif de type indiquant qu'une variable de ce type ne peut pas être modifiée (chapitre III). Il faut alors initialiser la variable à la déclaration.

continue : Provoque le rebouclage des boucles *while*, *do* et *for* (chapitre V–8). Voici un exemple de code qui affiche les valeurs de 1 à 10 autres que 4 et 7 en utilisant un *continue* :

```
int i;

for (i = 1; i <= 10; ++i)
{
      if (i == 4 || i == 7)
            continue;
```

```
      printf("%i\n", i);
}
```

On préférera le code suivant qui évite l'utilisation du *continue* :

```
int i;

for (i = 1; i <= 10; ++i)
{
      if (i != 4 && i != 7)
            printf("%i\n", i);
}
```

default : Choix par défaut d'un *switch* (voir le mot *switch*).

do : Boucle dans laquelle on entrera toujours au minimum une fois (chapitre V–6).

double : Type de donnée permettant de représenter des valeurs réelles (chapitres II–1.e et II–5). Il offre au moins 10 chiffres décimaux significatifs et un exposant allant au minimum de 10^{-37} à 10^{37}. Les constantes littérales réelles dans le code (comme 0.5 ou 3.1415) sont de ce type.

else : Permet de déterminer l'instruction à exécuter dans le cas où la condition d'un *if* est fausse (chapitre V–3).

enum : Type énuméré permettant de définir des constantes symboliques entières (chapitre XIX–2).

extern : Classe d'allocation par défaut des fonctions et des variables externes (chapitres VIII–6, IX–3, IX–4 et IX–8.a).

float : Type de donnée permettant de représenter des valeurs réelles (chapitres II–1.e et II–7). Il offre au moins 6 chiffres décimaux significatifs et un exposant allant au minimum de 10^{-37} à 10^{37}.

for : Boucle qui permet de regrouper l'initialisation, la condition et l'incrémentation dans la même instruction (chapitre V–5).

goto : Provoque un saut à une instruction étiquettée (chapitre V–11). Pour des raisons pédagogique, cette instruction est à éviter à tout prix en programmation structurée.

if : Permet de déterminer si une instruction doit être exécutée en fonction de l'évaluation d'une condition (chapitre V–2).

int : Type de donnée occupant au minimum 2 octets et permettant de représenter des valeurs entières (chapitres II–1.d, II–3, II–3.a et II–3.b). Les constantes littérales entières dans le code (comme 4 ou 100) sont de ce type.

long : Type de donnée occupant au minimum 4 octets et permettant de représenter des valeurs entières (chapitres II–1.d et II–3.c).

register : Classe d'allocation maintenant désuète (chapitres IX–3, IX–5.b et IX–8.b).

return : Permet d'interrompre l'exécution d'une fonction et de revenir à la fonction appelante en retournant (ou non) une valeur (chapitres V–9 et V–10).

short : Type de donnée occupant au minimum 16 bits et permettant de représenter des valeurs entières (chapitres II–1.d et II–3.f).

signed : Force une variable de type *char* à être signée (chapitre II–1.c).

sizeof : Opérateur retournant la taille en octets d'une variable ou d'un type lorsque c'est possible (chapitre IV–6).

static : Classe d'allocation indiquant qu'une variable doit être placée en mémoire permanente (chapitres IX–3, IX–5.c et IX–8.a). Ce mot permet aussi de rendre un identificateur confidentiel à une unité de compilation (chapitre VIII–8).

struct : Enregistrement formé de la juxtaposition d'objets de types pouvant être différents (chapitre XIX–3).

switch : Se rapprochant du *if*, cette instruction permet de choisir une instruction à effectuer en fonction de l'évaluation d'une expression (chapitre V–12).

typedef : Ersatz de définition de type permettant de créer des noms alias pour un nom de type (chapitre XVII–4).

union : Structure dont tous les membres sont implantés à la même adresse (chapitre XIX–4).

unsigned : Force une variable de type *char* à être non signée (chapitre II–1.c).

void : Type spécial indiquant qu'une fonction n'a pas de valeur de retour ou n'attend pas de paramètre (chapitre VIII–4.b). Il permet aussi de faire disparaître des valeurs dans un transtypage (chapitre IV–11.e). Finalement, il sert à créer des pointeurs génériques (chapitre XII).

volatile : Indique que le contenu d'une variable peut changer sous l'action d'un autre programme (chapitre III).

while : Boucle s'exécutant tant que la condition qu'elle contient est vraie (chapitre V–4).

3. MOTS RÉSERVÉS DU PRÉPROCESSEUR

*_DATE_* : Chaîne de caractères correspondant à la date de compilation du programme.

#define : Cette instruction permet de substituer un symbole par un autre (chapitre VI–1). Elle sert aussi à définir une macro-instruction paramétrée (chapitre VI–2).

defined : Opérateur retournant 1 si le symbole passé en paramètre est défini et 0 sinon (chapitre VI–4.c).

#elif : Accompagne un *#if*, un *#ifdef*, un *#ifndef* ou un autre *#elif*, elle correspond à l'instruction « *else if* » du préprocesseur (chapitre VI–4.c).

#else : Suit un *#if*, un *#ifdef*, un *#ifndef* ou un *#elif*, elle permet de déterminer les ligne à inclure dans le code quand l'expression évaluée est fausse (chapitre VI–4.c).

#endif : Terminaison d'un *#if*, *#ifdef* ou *#ifndef* (chapitre VI–4.c).

#error : Produit un message d'erreur à la compilation incluant l'argument passé.

*_FILE_* : Chaîne de caractères correspondant au nom du fichier compilé.

#if : Permet de vérifier si une valeur numérique est nulle (chapitre VI–4.c). Le résultat de l'évaluation permet de déterminer les lignes à inclure dans le code quand l'expression évaluée est vraie.

#ifdef : Permet de vérifier si un symbole a été défini (chapitre VI–4.c). Le résultat de l'évaluation permet de déterminer les lignes à inclure dans le code quand l'expression évaluée est vraie.

#ifndef : Permet de vérifier si un symbole n'a pas été défini (chapitre VI–4.c). Le résultat de l'évaluation permet de déterminer les lignes à inclure dans le code quand l'expression évaluée est vraie.

#include : Inclusion du contenu d'un fichier dans le fichier contenant l'instruction (VI–3).

#line : Averti le compilateur que le code source a été généré par un autre outil et indique la correspondance entre les nouvelles lignes de code et les lignes du code source écrit par l'usager.

*_LINE_* : Entier correspondant au numéro de ligne courant où se trouve l'instruction.

opérateur # : Permet de remplacer un argument formel par un argument effectif transformé en chaîne de caractères constante (chapitre VI–4.d).

opérateur ## : Permet de concaténer des symboles et des arguments formels (chapitre VI–4.e).

#pragma : Ajoute une nouvelle fonctionnalité au préprocesseur ou fournit une information spécifique à un compilateur.

*_STDC_* : La constante réelle 1.0 si le compilateur se conforme à la norme ISO.

*_STDC_VERSION_* : La constante 199409L si le compilateur est conforme à la norme C89 et 199901L s'il est conforme à la norme C99. Sinon, le symbole n'est pas défini.

*_TIME_* : Chaîne de caractères correspondant à l'heure de compilation du programme.

#undef : Annule une instruction *#define* (chapitre VI–4.b).