Programmation Procédurale

C3P

Vincent Aranega
vincent.aranega@univ-lille.fr

Université de Lille

Table of contents

Table of Contents

C'est quoi la programmation procédurale ?

Des idées ?

C'est quoi la programmation procédurale ?

La programmation procédurale est un paradigme de programmation qui utilise une approche linéaire ou descendante. Il s'appuie sur des procédures ou des sous-programmes pour effectuer des calculs et exprimer "comment" un programme doit résoudre un problème.

C'est quoi la programmation procédurale ?

- Assemblage de procédures/fonctions pour altérer l'état d'un système
- On utilise :
  - des données mutables
  - des effets de bord
- Impératif
- L'état de l'application passe par des données partagées
- Paradigme de programmation
  $\hookrightarrow$ il s'agit d'une façon de penser la construction d'un logiciel basée sur certains principes fondamentaux et définis

Quelles conséquences ?

En pratique

- On utilise beaucoup les boucles
- On utilise assez peu la récursivité
- Les données manipulées sont mutables

En résumé

La programmation procédurale tourne autour de données ainsi que de **fonctions** et **procédures** qui **altèrent** et manipulent ces données.

Histoire du C

C n'est pas un langage de "très haut niveau" et n'est pas spécialisé dans un domaine d'application particulier. Mais son absence de restrictions et sa généralité le rendent plus pratique et plus efficace pour de nombreuses tâches que des langages supposés plus puissants au prix d'un certains effort de développement.

- Langage développé par Dennis Ritchie en 1972 au laboratoire Bell-AT&T.
- Développé pour régler les problèmes des langages B, BCPL...etc
- Intialement dévelopé pour les OS UNIX, il hérite de beaucoup de fonctionnalités de B et BCPL.

1972 C traditionnel

1978 C K&R

1989 C Ainsi (C89)

1999 C99 proposé par un commité de standardisation

Table of Contents

Rappels – Pointeurs

- Les pointeurs sont des variables qui représentent des addresses mémoires.
- On les utilisent pour accéder et manipuler la mémoire.
- Si le type du pointeur est connu, il est possible d'effectuer de l'arithmétique sur pointeurs.

```c
void rebind(int ** var, int * a) {
  *var = a;
}

int main(void) {
  int a = 5;
  int b = 6;
  int * myvar = &a;
  printf("Value %d at addr-%p\n", *myvar, myvar);

  rebind(&myvar, &b);
  printf("Value %d at addr-%p\n", *myvar, myvar);
  return 0;
}

// Value 5 at addr-0x7ffcb0438b48
```

Rappels – Arithmétique de Pointeurs

- Toutes les opérations avec les pointeurs tiennent compte automatiquement du type et de la grandeur des objets pointés.

Exemple – Navigation dans un tableau

```
int A[10];
int *P;

P = A+9; P = A+10;
P = A+11; P = A-1;
```

Exemple – Soustraction de pointeurs

```
// P1 et P2 pointent sur deux elements du meme tableau
// Que dire du resultat ?
P1 - P2;
```

Pointeurs – Exercices

- Que représente chacune de ces expressions ?

```
int A;
&A

int B[];
B[i];  // Equivalent en arithmetique de pointeurs ?
&B[i];

int *P;
*P;
P;
P+i;
*(P+i);
```

Pointeurs – Exercices

- Écrire une fonction qui trie un tableau d'entiers passé en paramètres.
- Écrire une fonction qui retourne une vue d'un tableau rangé en ordre inverse sans modifier l'original et sans faire de copies de valeurs.

Rappels – Pointeurs génériques

- Les pointeurs génériques sont des pointeurs qui peuvent accueillir n'importe quel type d'adresse
- On ne peut pas accéder à la valeur d'un pointeur générique directement sans l'avoir "casté" au préalable
- On ne peut pas faire d'arithmétique sur pointeurs génériques
- Pour manipuler un pointeur générique, il est nécessaire de le transformer en un type connu.

```c
void print_addr(void * o) {
  printf("my address is %p\n", o);
}

int main(void) {
  int a = 4;
  float b = 1.0;

  print_addr(&a);
  print_addr(&b);
  return 0;
}
```

Rappels – Pointeurs de fonctions

- Un pointeur de fonction est un pointeur vers une fonction.
- On peut passer une fonction en paramètre ou en retour d'une autre.
- On déclare un pointeur de fonction par rapport à la signature de la fonction qu'il manipule.

```
void add_int(void * a, void * b) {
  printf("%d\n", *(int *)a + *(int *)b);
}

void add_float(void * a, void * b) {
  printf("%f\n", *(float *)a + *(float *)b);
}

void add(void * a, void * b, void add(void *, void *)) {
  add(a, b);
}
```

Pointeurs – Exercices

- Comment trier une liste selon différents critères non connue à l'avance ?
- Écrire un programme capable d'adapter un affichage en fonction d'un type de structure sans utiliser de conditions ?

Rappels – Structures

- Définition des champs et de leurs types
- Possibilité d'avoir des structures récursives

```
typedef struct _list list;
struct _list {
  void * value;
  list * next;
};

void add_header(list ** head, void * value) {
  list * new = malloc(sizeof(list));
  new->value = value;
  new->next = head;
  *head = next;
}
```

Pointeurs – Exercices

- Proposez un modèle permettant d'effectuer de l'OOP sans héritage.

Rappels – Lecture dans fichiers

La lecture dans un fichier se fait par effet de bords, on ouvre un fichier dans un mode particulier, on vérifie qu'il est bien ouvert, on lit/écrit à l'intérieur une ou plusieurs informations et on ferme le fichier à la fin.

```c
#include <stdio.h>

int main(void) {
  int rinteger;
  FILE *fp = fopen("myfile", "r");
  if (fp == NULL) {
    fprintf(stderr, "Impossible to open the file");
    return 1;
  }
  fread(&rinteger, sizeof(int), 1, fp);
  printf("I read %d\n", rinteger);
  return 0;
}
```

Table of Contents

Framework check

Il existe plusieurs frameworks de tests, un des plus utilisés est check. Une suite de test en check est un programme qui définit les cas de tests, les suites de tests et un moyen de les lancer. Il n'y a pas autant de magie que dans des frameworks tels que PyTest ou JUnit.

```
#include <check.h>
#include <strings.h>

START_TEST (dumb_check)
{
  char test[] = "This is a string";

  ck_assert_str_eq("This is a string", test);
  ck_assert_int_eq(16, strlen(test));
}
END_TEST
```

Framework check – Suite de tests

Une fois que le test case est définit, il faut le rajouter à une suite de tests. Une suite de tests enregistre différent cas de tests pour une thématique en particulier.

```
Suite * test_about_strings(void)
{
  Suite *s = suite_create("Suite de tests pour str");
  TCase *tc_core = tcase_create("Core");
  tcase_add_test(tc_core, dumb_check);
  suite_add_tcase(s, tc_core);
  return s;
}
```

Framework check – Runner

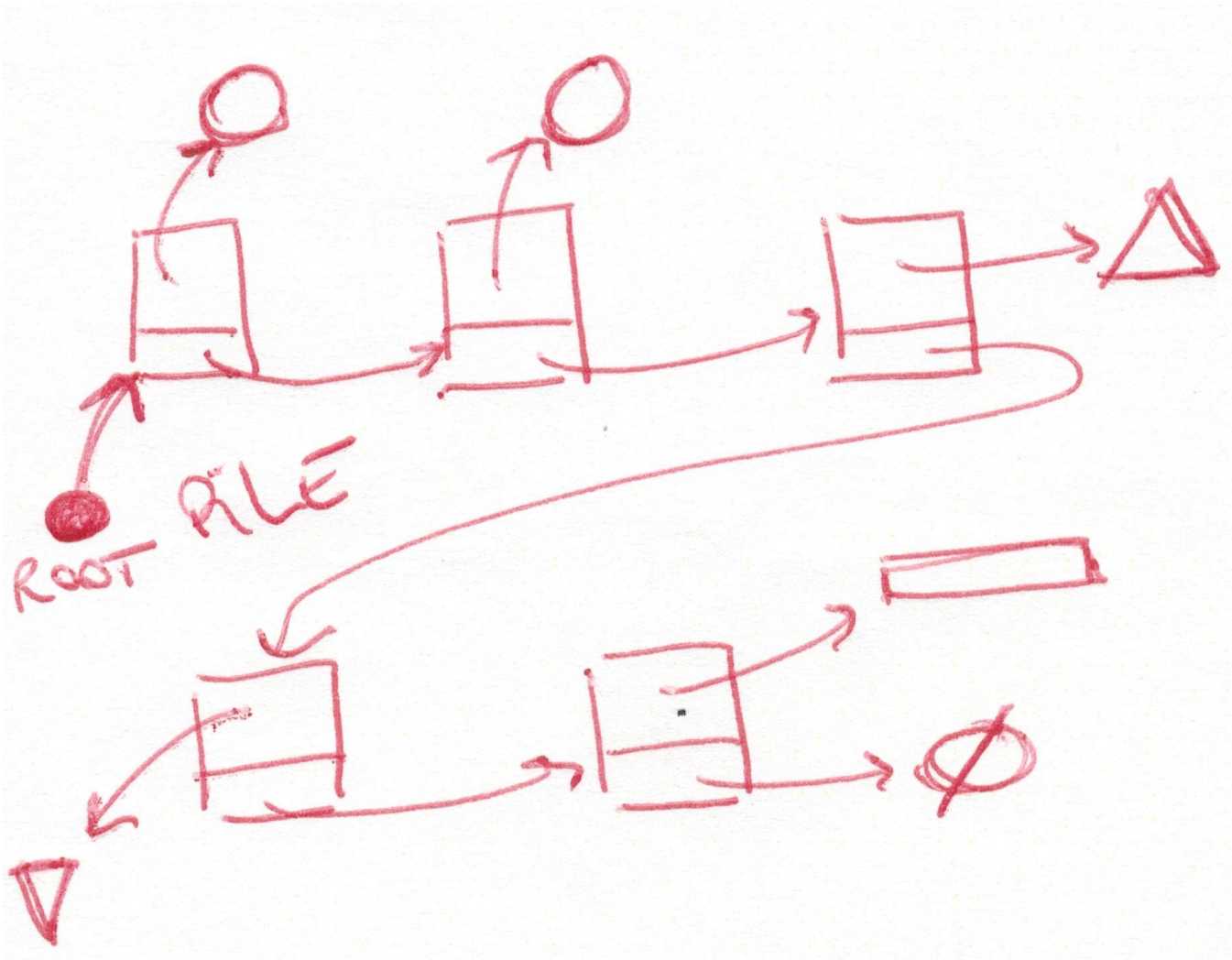
Une fois la suite de test créée, il est nécessaire de fournir un moyen de lancer la suite de tests. C'est le travail du runner. En check, un runner très simple pourrait être définit comme suit :

```
int main(void)
{
  int number_failed;
  Suite *s = test_about_strings();
  SRunner *sr = srunner_create(s);

  srunner_run_all(sr, CK_NORMAL);
  number_failed = srunner_ntests_failed(sr);
  srunner_free(sr);
  return (number_failed == 0) ? 0 : 1;
}
```

Table of Contents

Pile à partir d'une liste chainée

- Comment représenter le système ?
- Quels sont les états du système à conserver ?



Pile à partir d'une liste chainée – Tests

```
START_TEST (check_empty) {
  stack *s;
  init_stack(&s);
  ck_assert(is_empty(s));
  ck_assert_int_eq(0, size_stack(s));
} END_TEST
```

Pile à partir d'une liste chainée – Implem

```c
typedef struct _stack stack;
struct _stack {
  void * value;
  struct _stack * next;
};

void init_stack(stack ** s) {
  *s = NULL;
}

int is_empty(stack * s) {
  return s == NULL;
}

int size_stack(stack * s) {
  int i = 0;
  for (stack * current = s;
       !is_empty(current);
       current = current->next, i++);
  return i;
}
```

Pile à partir d'une liste chainée – Tests

```
START_TEST (check_peek_push) {
  stack *s;
  int x = 4;
  init_stack(&s);
  push_stack(&s, &x);
  ck_assert_int_eq(4, * (int *)peek_stack(s));

  char c[] = "This is a string";
  push_stack(&s, &c);
  ck_assert_str_eq("This is a string", peek_stack(s));

  ck_assert(!is_empty(s));
  ck_assert_int_eq(2, size_stack(s));
} END_TEST
```

Pile à partir d'une liste chainée – Implem

```
void push_stack(stack ** s, void * value) {
  stack * new = malloc(sizeof(stack));
  new->value = value;
  new->next = *s;
  *s = new;
}

void * peek_stack(stack * s) {
  return s->value;
}
```

Pile à partir d'une liste chainée – Tests

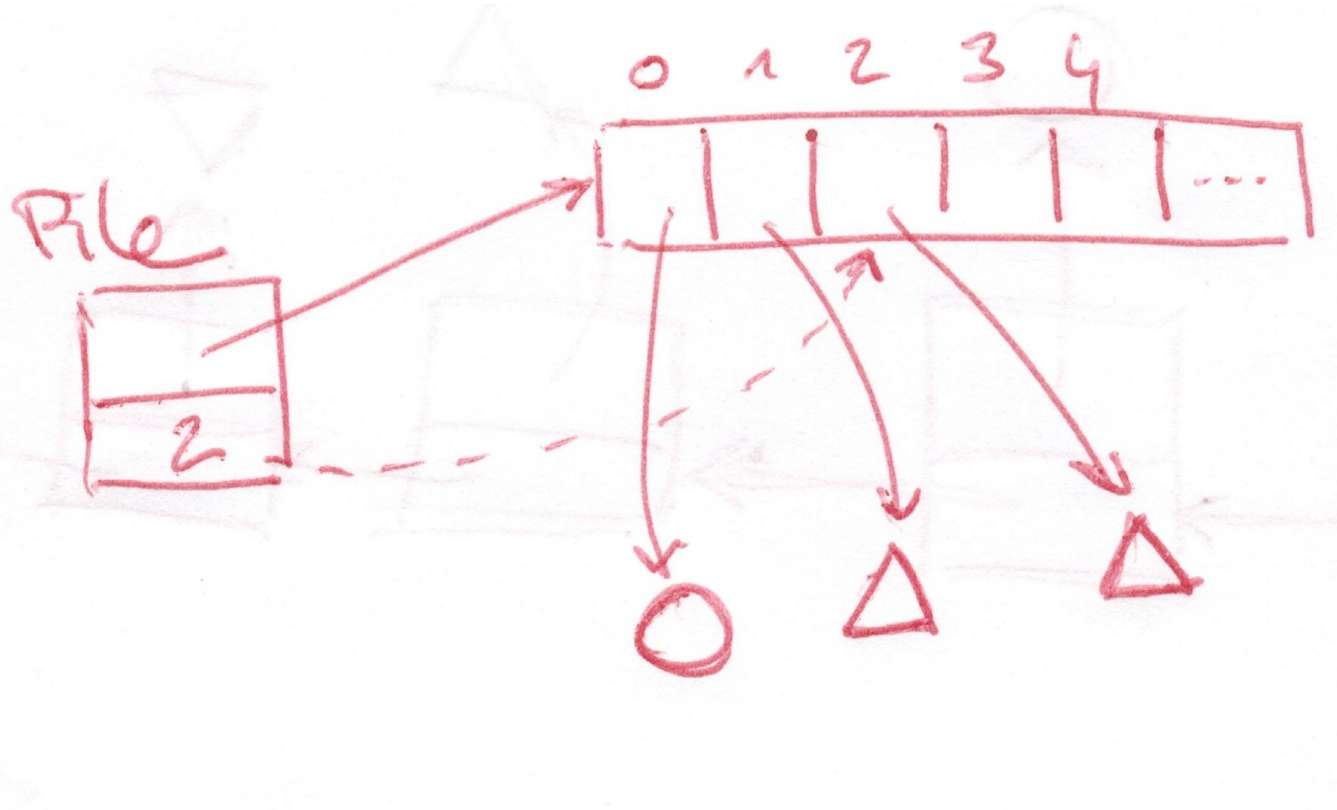
```
START_TEST (check_pop) {
  stack *s;
  init_stack(&s);

  int x = 4;
  char c[] = "This is a string";
  push_stack(&s, &x);
  push_stack(&s, &c);
  ck_assert_str_eq("This is a string", pop_stack(&s));
  ck_assert_int_eq(1, size_stack(s));
  ck_assert_int_eq(4, * (int *)pop_stack(&s));
  ck_assert_int_eq(0, size_stack(s));
  ck_assert_ptr_null(pop_stack(&s));
  ck_assert_int_eq(0, size_stack(s));
  destroy_stack(&s);
} END_TEST

TSUITE(basic_stack, "Basic stack testing",
       check_empty, check_peek_push, check_pop)

RUN_ALL(basic_stack)
```

Pile à partir d'une liste chainée – Implem

```
void * pop_stack(stack **s) {
  if (is_empty(*s)) return NULL;
  stack * old = *s;
  void * value = old->value;
  *s = old->next;
  free(old);
  return value;
}

void destroy_stack(stack ** s) {
  if (is_empty(*s)) return;
  while (pop_stack(s) != NULL);
}
```

Pile à partir d'un tableau

- Comment représenter le système ?
- Quels sont les états du système à conserver ?



Pile à partir d'un tableau

- Comment représenter le système ?
- Quels sont les états du système à conserver ?

Pile à partir d'un tableau

- Comment représenter le système ?
- Quels sont les états du système à conserver ?

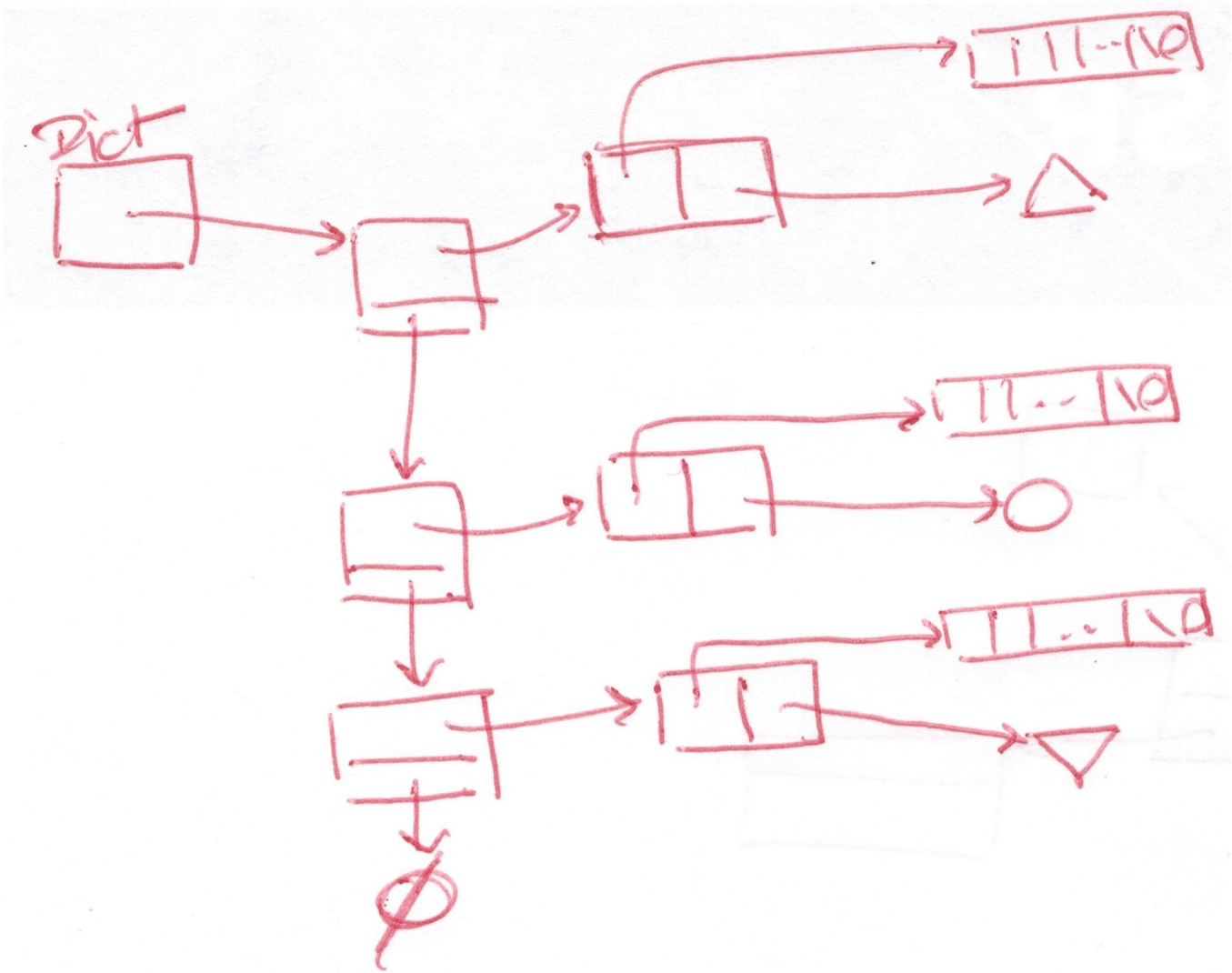
```c
#define STACK_SIZE 5

typedef struct {
  void * tab[STACK_SIZE];
  int top;
} stack;

void init_stack(stack * s) {
  s->top = -1;
}

int is_empty(stack * s) {
  return s->top == -1;
}
```

Pile à partir d'un tableau

```c
#define STACK_FULL  (-1)
#define STACK_EMPTY NULL

int push(stack * s, void * value) {
  if (s->top == STACK_SIZE - 1) return STACK_FULL;
  s->tab[++s->top] = value;
  return 0;
}

void * peek(stack * s) {
  if (is_empty(s)) return STACK_EMPTY;
  return s->tab[s->top];
}

void * pop(stack * s) {
  if (is_empty(s)) return STACK_EMPTY;
  void * value = s->tab[s->top];
  s->top--;
  return value;
}
```

Dictionnaire

- On considère ici un dico très simple
- Pas de remplacement ni de suppression de clefs



Dictionnaire

- On considère ici un dico très simple
- Pas de remplacement ni de suppression de clefs

Dictionnaire

- On considère ici un dico très simple
- Pas de remplacement ni de suppression de clefs

```
typedef struct {
  stack * stack;
} dict;

typedef struct {
  char * key;
  void * value;
} entry;
```

Dictionnaire

```
dict * create_dict() {
  dict * d = malloc(sizeof(dict));
  init_stack(&(d->stack));
  return d;
}

void put(char * key, void * value, dict * d) {
  entry * e = malloc(sizeof(entry));
  e->key = malloc(sizeof(char) * strlen(key));
  e->value = value;
  push_stack(&(d->stack), e);
}

void * get(char * key, dict * d) {
  stack * s = d->stack;
  while (s != NULL) {
    entry * e = (entry *)s->value;
    if (strcmp(e->key, key)) return e->value;
    s = s->next;
  }
  return NOT_FOUND;
}
```

Dictionnaire

```
void destroy_entry(entry * e) {
  free(e->key); free(e);
}

void destroy_dict(dict * d) {
  stack * s = d->stack;
  while (s != NULL) {
    destroy_entry(s->value);
    s = s->next;
  }
  destroy_stack(&(d->stack)); free(d);
}

int main(void)
{
  dict *d = create_dict();
  int a = 3;
  put("a", &a, d);
  int x = * (int *)get("a", d);
  destroy_dict(d);
  return 0;
}
```

Table of Contents

Exemples en Pascal

```
procedure NumeroFoisDix( numero : integer ) ;
var j : integer;
begin
    j := numero * 10 ;
    println('j␣=␣', j) ;
end ;

function NumeroPlusCinq( numero : integer ) : integ
var j : integer ;
begin
    j := numero + 5 ;
    NumeroPlusCinq := j ;
end ;
```

Exemples en assembleur 6502

```
MULT10   ASL          ;multiply by 2
         STA TEMP     ;temp store in TEMP
         ASL          ;again multiply by 2 (*4)
         ASL          ;again multiply by 2 (*8)
         CLC
         ADC TEMP     ;as result, A = x*8 + x*2
         RTS

TEMP     .byte 0
```

Exemples en Python