
Triangle de Pascal

1			
1	1		
1	2	1	
1	3	3	1

Bastien Delseny

Table des matières

1	Compilation et génération du fichier exécutable	1
1.1	Compilation de triangle	1
1.2	Redirection de la sortie standard	1
1.3	Redirection de l'entrée standard	1
1.4	Tube et chaînage de commandes	2
2	Lancement de l'interprète PostScript par exec	2
3	Lancement de gs par fork et exec	4
3.1	Vérification de la taille de police et du nombre de lignes	4
3.2	Une histoire de père et de fils	4

1 Compilation et génération du fichier exécutable

1.1 Compilation de triangle

L'exécution du programme `triangle` permet d'obtenir les instructions en PostScript pour la construction d'un triangle de Pascal de taille de police et de nombre de lignes voulu. Pour exécuter ce programme il suffit d'exécuter la commande `>./triangle 12 4` ce qui donnera les instructions en PostScript pour créer un triangle de Pascal de taille de police 12 et de 4 lignes (code 1).

Listing 1 – `./triangle 12 4`

```
newpath 24 245 moveto 38 245 lineto 38 260 lineto stroke newpath  
  ↪ 24 245 moveto 24 260 lineto 38 260 lineto stroke /Courier  
  ↪ findfont 12 scalefont setfont newpath 25 248 moveto (1)  
  ↪ show  
newpath 24 230 moveto 38 230 lineto 38 245 lineto stroke newpath  
  ↪ 24 230 moveto 24 245 lineto 38 245 lineto stroke /Courier  
  ↪ findfont 12 scalefont setfont newpath 25 233 moveto (1)  
  ↪ show  
etc ...
```

1.2 Redirection de la sortie standard

La commande `>./triangle 12 4 > triangle_pascal_12_4.ps` permet de rediriger la sortie standard vers le fichier `triangle_pascal_12_4.ps` qui contient le code PostScript 1. À l'aide de la commande `> cat triangle_pascal_12_4.ps` on affiche dans le terminal le code PostScript (code 1) enregistré dans `triangle_pascal_12_4.ps`. À l'aide de la commande `>gv triangle_pascal_12_4.ps` l'interprète de langage PostScript nous permet d'observer la figure 1.

1.3 Redirection de l'entrée standard

L'exécution de la commande `> tr "bc" "BC" < Makefile` permet de mettre en majuscule les lettres c et b du fichier `Makefile`.

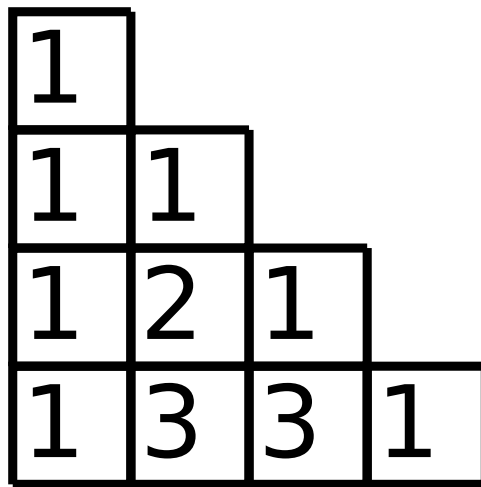


FIGURE 1 – Triangle de Pascal de taille de police 12 de 4 lignes.

1.4 Tube et chaînage de commandes

Lorsque l'on exécute la commande `> date > d` on redirige la sortie de l'exécution de la commande `> date` vers le fichier `d`. L'exécution de la commande `> tr ":" "_" < d` permet de récupérer ce que contient le fichier `d` afin de remplacer les `:` par des `_`. À l'aide de l'utilisation d'un tube de redirection il est possible d'effectuer la même manoeuvre sans passer par un fichier intermédiaire et afficher le résultat directement dans le terminal. C'est ce que permet de faire la commande `date | tr ":" "_"`.

Ainsi en utilisant les commandes `2` on obtient la même figure `1` qu'en effectuant les démarche de la section `1.2` qui sera affichée par un interpréteur de PostScript.

Listing 2 – Tubes de redirection

```
> ./triangle 12 4 | gs -sDEVICE=x11 -q
> ./triangle_12_4 | gv -
```

2 Lancement de l'interprète PostScript par exec

Les exécutables des commandes `gv` et `gs` se trouvent dans les répertoire `/usr/bin/gv` et `/usr/bin/gs`, chemin qu'il a été possible de trouver grâce à la commande `which`. Ainsi exécuter la commande

>/usr/bin/gv triangle.ps revient au même que d'exécuter la commande >gv triangle.ps.

Afin d'exécuter directement l'interpreteur de PostScript lors de l'exécution du programme il faut modifier le code source de `affiche_triangle`. Il faudra commencer par initialiser deux variables. Un texte contenant le chemin de la commande de l'interpreteur PostScript à utiliser. Ainsi qu'un tableau de texte contenant la commande ainsi que ces arguments. Le code nécessaire pour l'interpreteur `gv` et celui pour `gs` est ici : 3.

Listing 3 – Initialisation des variables pour exec de gv et gs

```
/*      Arguments pour gv      */
char nomGV [] = "/usr/bin/gv";
char *argumentsGV [] = {"/usr/bin/gv", "triangle.ps", NULL};

/*      Arguments pour gs      */
char nomGS [] = "/usr/bin/gs";
char *argumentsGS [] = {"/usr/bin/gs", "-q", "-sDEVICE=x11", "
    ↪ triangle.ps", NULL};
```

Ensuite il faudra ajouter la commande `execve(...)` correspondante à l'interpreteur choisi à la fin du `main()` (code 4)

Listing 4 – Appel de la fonction execve() pour gv et gs

```
/*      Execution de gv      */
execve(nomGV, argumentsGV, envp);

/*      Execution de gs      */
execve(nomGS, argumentsGS, envp);
```

Puis il faut recompiler le code avec >make. Il suffira d'exécuter le programme à l'aide de la commande >./triangle 12 4 > triangle.ps pour obtenir le même résultat que pour la section 1.4. On peut en plus rediriger la sortie standard d'erreur dans un fichier à l'aide de l'option 2> fichier.log comme suit >./triangle 12 4 > triangle.ps 2> triangle.log.

3 Lancement de gs par fork et exec

3.1 Vérification de la taille de police et du nombre de lignes

Afin de vérifier si la taille de la police et si le nombre de ligne sont correctes on peut ajouter le code suivant 5 dans le code source de `affiche_triangle.c`

Listing 5 – Vérification de la taille de police comprise entre 8 et 24, et du nombre de lignes compris entre 1 et 40.

```
if (taille_police < 8 || taille_police > 24){
    fprintf(stderr, "Taille_de_police_incorrecte ,_valeur_
    ↪ entre_8_et_24_attendue\n");
    exit(1);
}
if (nb_lignes < 1 || nb_lignes > MAX_LIGNES){ // MAX_LIGNES = 40
    fprintf(stderr, "Nombre_de_lignes_incorrecte ,_valeur_
    ↪ entre_1_et_%d_attendue\n", MAX_LIGNES);
    exit(1);
}
```

3.2 Une histoire de père et de fils

La fonction C `fork()` permet de créer un fils. Il faut commencer par initialiser deux variables dans `affiche_triangle.ps`. Une variable qui permettra de récupérer le pid du fils grâce à la commande `pid_t p`; Il faut aussi une variable qui permettra de récupérer le code de retour, code d'erreur ou de réussite de la fin d'exécution du fils, grâce à la commande `int retour`; Afin que le processus fils exécute `gv` ou `gs` il faut procéder comme indiquer dans le code 6.

Listing 6 – Création d'un fils qui execute la fonction `execve` permettant la lecture du fichier PostScript par un interpréteur de PostScript.

```
/* Generation des triangles par le pere */
sortie = "stdout";
taille_triangle = nb_lignes;
```

```
postscript_triangle (taille_police);
sleep (3);

/* Creation d'un processus fils */
p = fork();

/* Verification de la creation d'un fils a l'aide de la commande
   ↪ fork */
if(p < 0) {
    fprintf(stderr, "Luke_(fils)_n'a_pas_pu_etre_creer_par_
        ↪ fork\nVador_(pere)_est_triste\n");
    exit(1);
}

/* Fils creer plus qu'a attendre qu'il affiche les triangles*/
if(p != 0){
    /* Attendre la terminaison du fils */
    wait(&retour);
    fprintf(stderr, "Generation_et_affichage_du_triangle_de_
        ↪ Pascal_termine\n");
} else {
    /* Execution de l'interpreteur de PostScript */
    execve(nomGV, argumentsGV, envp);
    //execve(nomGS, argumentsGS, envp);
}
```