

voici comment procéder :

1. D'abord, créez un fichier lexer avec Flex (par exemple `dessin.1`) pour reconnaître les tokens :

```
c
%{
#include <string.h>
#include "dessin.tab.h"
%}

%%
"Draw"      { return DRAW; }
"Circle"    { return CIRCLE; }
"Rectangle" { return RECTANGLE; }
"Triangle"  { return TRIANGLE; }
"rouge"     { yyval.couleur = 2; return COULEUR; }
"vert"      { yyval.couleur = 3; return COULEUR; }
"bleu"      { yyval.couleur = 5; return COULEUR; }
"noir"      { yyval.couleur = 1; return COULEUR; }
"jaune"     { yyval.couleur = 4; return COULEUR; }
"magenta"   { yyval.couleur = 6; return COULEUR; }
"cyan"      { yyval.couleur = 7; return COULEUR; }
"blanc"     { yyval.couleur = 8; return COULEUR; }
"[0-9]+"    { yyval.nombre = atoi(yytext); return NOMBRE; }
"("        { return OUVIR_PAR; }
")"        { return FERMER_PAR; }
","        { return VIRGULE; }
"[\t\n]"    { /* ignorer les espaces */ }
.          { fprintf(stderr, "Caractère non reconnu: %s\n", yytext); }
%%
```

2. Ensuite, créez un fichier parser avec Bison (par exemple `dessin.y`) :

```
c
%{
#include <stdio.h>
#include <stdlib.h>

extern int yylex();
extern int yyparse();
extern FILE *yyin;

void yyerror(const char *s);
void genererCode(char* forme, int taille, int couleur);
%}

%union {
    int nombre;
    int couleur;
}

%token DRAW CIRCLE RECTANGLE TRIANGLE
%token OUVIR_PAR FERMER_PAR VIRGULE
%token <nombre> NOMBRE
%token <couleur> COULEUR

%%

programme:
    instruction
    | programme instruction
    ;
```

```

%{
#include <stdio.h>
#include <stdlib.h>

extern int yylex();
extern int yyparse();
extern FILE *yyin;

void yyerror(const char *s);
void genererCode(char* forme, int taille, int couleur);
%}

%union {
    int nombre;
    int couleur;
}

%token DRAW CIRCLE RECTANGLE TRIANGLE
%token OUVRIR_PAR FERMER_PAR VIRGULE
%token <nombre> NOMBRE
%token <couleur> COULEUR

%%

programme:
    instruction
    | programme instruction
    ;

instruction:
    DRAW CIRCLE OUVRIR_PAR NOMBRE VIRGULE COULEUR FERMER_PAR {
        genererCode("cercle", $4, $6);
    }
    | DRAW RECTANGLE OUVRIR_PAR NOMBRE VIRGULE COULEUR FERMER_PAR {
        genererCode("rectangle", $4, $6);
    }
    | DRAW TRIANGLE OUVRIR_PAR NOMBRE VIRGULE COULEUR FERMER_PAR {
        genererCode("triangle", $4, $6);
    }
    ;

%%

void yyerror(const char *s) {
    fprintf(stderr, "Erreur de parsing: %s\n", s);
    exit(1);
}

```

```

void genererCode(char* forme, int taille, int couleur) {
    FILE *fichier = fopen("dessin_genere.c", "w");

    // Écrire l'en-tête du fichier
    fprintf(fichier, "#include <SDL2/SDL.h>\n");
    fprintf(fichier, "#include <stdio.h>\n");
    fprintf(fichier, "#include <stdlib.h>\n");
    fprintf(fichier, "#include <math.h>\n\n");

    // Copier les fonctions utilitaires
    fprintf(fichier, "void dessinerRectangle(SDL_Renderer *renderer, SDL_Color couleur) {\n");
    fprintf(fichier, "    SDL_SetRenderDrawColor(renderer, couleur.r, couleur.g, couleur.b,\n");
255);\n");
    fprintf(fichier, "    SDL_Rect rect = {200, 150, 300, 200};\n");
    fprintf(fichier, "    SDL_RenderFillRect(renderer, &rect);\n");
    fprintf(fichier, "}\n\n");

    fprintf(fichier, "void dessinerTriangle(SDL_Renderer *renderer, SDL_Color couleur) {\n");
    fprintf(fichier, "    SDL_SetRenderDrawColor(renderer, couleur.r, couleur.g, couleur.b,\n");
255);\n");
    fprintf(fichier, "    SDL_RenderDrawLine(renderer, 400, 100, 200, 400);\n");
    fprintf(fichier, "    SDL_RenderDrawLine(renderer, 200, 400, 600, 400);\n");
    fprintf(fichier, "    SDL_RenderDrawLine(renderer, 600, 400, 400, 100);\n");
    fprintf(fichier, "}\n\n");

    fprintf(fichier, "void dessinerCercle(SDL_Renderer *renderer, SDL_Color couleur, int cx,\n");
int cy, int rayon) {\n");
    fprintf(fichier, "    SDL_SetRenderDrawColor(renderer, couleur.r, couleur.g, couleur.b,\n");
255);\n");
    fprintf(fichier, "    for (int w = 0; w < rayon * 2; w++) {\n");
    fprintf(fichier, "        for (int h = 0; h < rayon * 2; h++) {\n");
    fprintf(fichier, "            int dx = rayon - w;\n");
    fprintf(fichier, "            int dy = rayon - h;\n");
    fprintf(fichier, "            if (dx * dx + dy * dy <= rayon * rayon) {\n");
    fprintf(fichier, "                SDL_RenderDrawPoint(renderer, cx + dx, cy + dy);\n");
    fprintf(fichier, "            }\n");
    fprintf(fichier, "        }\n");
    fprintf(fichier, "    }\n");
    fprintf(fichier, "}\n\n");

    fprintf(fichier, "SDL_Color choisirCouleur(int choix) {\n");
    fprintf(fichier, "    SDL_Color couleurs[] = {\n");
    fprintf(fichier, "        {0, 0, 0},    // noir\n");
    fprintf(fichier, "        {255, 0, 0},  // rouge\n");
    fprintf(fichier, "        {0, 255, 0},  // vert\n");
    fprintf(fichier, "        {255, 255, 0}, // jaune\n");
    fprintf(fichier, "        {0, 0, 255},  // bleu\n");
    fprintf(fichier, "        {255, 0, 255}, // magenta\n");

```

```

fprintf(fichier, "    {0, 255, 255}, // cyan\n");
fprintf(fichier, "    {255, 255, 255} // blanc\n");
fprintf(fichier, "    };\n");
fprintf(fichier, "    if (choix >= 1 && choix <= 8)\n");
fprintf(fichier, "        return couleurs[choix - 1];\n");
fprintf(fichier, "    else\n");
fprintf(fichier, "        return (SDL_Color){255, 255, 255};\n");
fprintf(fichier, "    }\n\n");

// Fonction principale personnalisée en fonction des paramètres
fprintf(fichier, "int main() {\n");
fprintf(fichier, "    if (SDL_Init(SDL_INIT_VIDEO) != 0) {\n");
fprintf(fichier, "        printf(\"Erreur SDL: %s\n\", SDL_GetError());\n");
fprintf(fichier, "        return 1;\n");
fprintf(fichier, "    }\n\n");

    fprintf(fichier, "    SDL_Window *window = SDL_CreateWindow(\"Dessin
Automatique\", \n");
    fprintf(fichier, "        SDL_WINDOWPOS_CENTERED,
SDL_WINDOWPOS_CENTERED, 800, 600, 0);\n\n");

    fprintf(fichier, "    SDL_Renderer *renderer = SDL_CreateRenderer(window, -1,
SDL_RENDERER_ACCELERATED);\n");
    fprintf(fichier, "    SDL_SetRenderDrawColor(renderer, 255, 255, 255, 255);\n");
    fprintf(fichier, "    SDL_RenderClear(renderer);\n\n");

    fprintf(fichier, "    SDL_Color couleur = choisirCouleur(%d);\n\n", couleur);

// Code spécifique à la forme
if (strcmp(forme, "cercle") == 0) {
    fprintf(fichier, "    dessinerCercle(renderer, couleur, 400, 300, %d * 20);\n", taille);
} else if (strcmp(forme, "rectangle") == 0) {
    fprintf(fichier, "    dessinerRectangle(renderer, couleur);\n");
} else if (strcmp(forme, "triangle") == 0) {
    fprintf(fichier, "    dessinerTriangle(renderer, couleur);\n");
}

    fprintf(fichier, "\n    SDL_RenderPresent(renderer);\n\n");

    fprintf(fichier, "    SDL_Event e;\n");
    fprintf(fichier, "    int quit = 0;\n");
    fprintf(fichier, "    while (!quit) {\n");
    fprintf(fichier, "        while (SDL_PollEvent(&e)) {\n");
    fprintf(fichier, "            if (e.type == SDL_QUIT)\n");
    fprintf(fichier, "                quit = 1;\n");
    fprintf(fichier, "        }\n");
    fprintf(fichier, "    }\n\n");

```

```

fprintf(fichier, "    SDL_DestroyRenderer(renderer);\n");
fprintf(fichier, "    SDL_DestroyWindow(window);\n");
fprintf(fichier, "    SDL_Quit();\n\n");

fprintf(fichier, "    return 0;\n");
fprintf(fichier, "}\n");

fclose(fichier);
printf("Code généré avec succès dans 'dessin_genere.c'\n");
}

int main(int argc, char **argv) {
    if (argc > 1) {
        FILE *f = fopen(argv[1], "r");
        if (!f) {
            perror(argv[1]);
            return 1;
        }
        yyin = f;
    }

    yyparse();
    return 0;
}

```

dessin.tab.c dessin.tab.h exemple.dessin makefile
dessin.tab.c dessin.y lex.yy.c

Il semble que vous rencontrez deux problèmes avec la compilation :

1. Un avertissement concernant la fonction `strcmp` qui n'est pas déclarée correctement dans le fichier `dessin.y`
2. Une erreur plus importante : le linker ne trouve pas la bibliothèque `-ly` (libbison)

Voici comment résoudre ces problèmes :

1. Ajouter l'include pour `string.h`

Modifiez votre fichier `dessin.y` pour ajouter l'include manquant au début :

```

c
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h> // Ajoutez cette ligne

extern int yylex();
extern int yyparse();
extern FILE *yyin;

void yyerror(const char *s);
void genererCode(char* forme, int taille, int couleur);
%}

```

2. Modifier le Makefile pour résoudre le problème de bibliothèque

Sur certaines distributions Linux, la bibliothèque `liby` n'est pas installée par défaut ou porte un autre nom. Modifiez votre `makefile` comme suit :

```
makefile

all: dessin_compiler

dessin_compiler: lex.yy.c dessin.tab.c
    gcc -o dessin_compiler lex.yy.c dessin.tab.c -lfl

lex.yy.c: dessin.l dessin.tab.h
    flex dessin.l

dessin.tab.c dessin.tab.h: dessin.y
    bison -d dessin.y

clean:
    rm -f dessin_compiler lex.yy.c dessin.tab.c dessin.tab.h dessin_genere.c
```

J'ai retiré le flag `-ly` car il semble qu'il ne soit pas disponible sur votre système. Sur les systèmes Ubuntu/Debian, vous pourriez avoir besoin d'installer le paquet `libbison-dev` :

```
bash

sudo apt-get install libbison-dev
```

Si vous préférez continuer sans installer ce paquet, la modification du Makefile devrait suffire, car Bison génère du code C autonome qui ne nécessite pas forcément de bibliothèque à l'exécution.

Une fois ces modifications effectuées, essayez à nouveau de compiler votre projet :

```
bash

make clean
make
./dessin_compiler exemple.dessin
```

Si tout se passe bien, vous devriez obtenir le fichier `dessin_genere.c` que vous pourrez ensuite compiler avec:

Une fois ces modifications effectuées, essayez à nouveau de compiler votre projet :

```
bash

make clean
make
./dessin_compiler exemple.dessin
```

Si tout se passe bien, vous devriez obtenir le fichier `dessin_genere.c` que vous pourrez ensuite compiler avec:

```
bash

gcc -o dessin_executable dessin_genere.c -lSDL2 -lm
```

Puis exécuter :

```
bash

./dessin_executable
```

① Claude n'a pas encore la capacité d'exécuter le code qu'il génère.

Pour utiliser ces fichiers :

1. Créez un fichier `exemple.dessin` avec vos commandes de dessin :

```
Draw Rectangle(10,noir)
```

2. Compilez avec le Makefile :

```
bash
make clean
make
```

3. Exécutez votre compilateur sur le fichier d'entrée :

```
bash
./dessin_compiler exemple.dessin
```

4. Compilez le code C généré :

```
bash
gcc -o dessin_executable dessin_genere.c -lSDL2 -lm
```

5. Exécutez le programme :

```
bash
./dessin_executable
```

Le programme traitera maintenant correctement les différentes formes, tailles et couleurs spécifiées dans vos commandes.

🔔 Claude n'a pas encore la capacité d'exécuter le code qu'il génère.



Réessayer ▾