プログラミング演習 レポート

アナログ時計

提出期限　2020年11月24日 17:00
提出日 2020年11月24日

組番号　408
学籍番号　17406

氏名　金澤雄大

1　目的

後期のプログラミング演習で学習した内容の理解度を確認するためにアナログ時計のアプリケーションを作成する.

2　実行環境

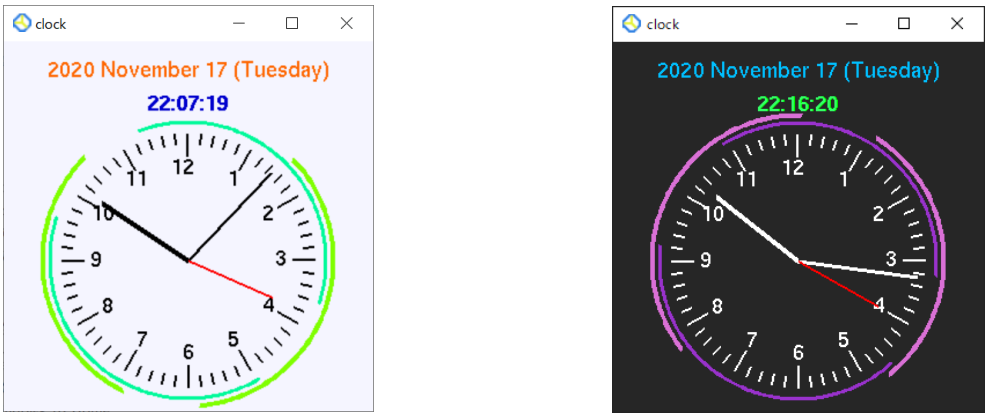
実行環境を 1 に示す.gcc とは「GNU Compiler Collection」の略称で,GNU プロジェクトが公開しているコンパイラのことである. make は Makefile にプログラムのコンパイルやリンクの方法を指示することで, コンパイルを簡単に行うことができるツールのことである. make を用いることは,gcc コンパイル時に, 長いオプションを入力しなくてよい, ファイルの更新を取得して必要なものだけをコンパイルしてくれるという利点がある.

表 1: 実行環境

CPU	Intel(R) Core(TM) i7-6500U 2.50GHz
メモリ	16.0GB DDR4
OS	Microsoft Windows 10 Home
gcc	version 9.3.0
make	version 4.3

3　アプリケーションの説明

図 1 に示すアナログ時計のアプリケーションを作成した. 時計の表示にはライトモードとダークモードの 2 種類がある. ライトモードとは白を基調とした画面表示のことであり, ダークモードは黒を基調とした画面表示のことである. 図 11a はライトモードのときの時計の表示例, 図 11b はダークモードのときの時計の表示例である. モードの切り替えは, 時計アプリのウィンドウをマウスで左クリックすることで行うことができる.



(a) ライトモードの時計　　(b) ダークモードの時計

図 1: 時計のアプリケーション

このアプリでは次に示すものを画面に描画する. アナログ時計の縁は内側と外側があり, 回転している. 回転方向は外側が時計回り, 内側が反時計周りである.

- アナログ時計
- アナログ時計の縁
- 年, 月, 日, 曜日, 時, 分, 秒の文字列

また, レポートでは伝わらないが, ライトモードとダークモードでは画面に描画しているアナログ時計および文字列の色が異なる. 表 2 に 2 つのモードにおける色の設定を示す.

表 2: 色の設定

描画内容	ライトモードでの色	ダークモードでの色
背景色	白	黒
年 月 日 (曜日)	オレンジ	ライトブルー
時:分:秒	青	緑
時計の針 (分針, 時針) およびインデックス	黒	白
時計の針 (秒針)	赤	赤
時計の縁 (内側)	スプリンググリーン	パープル
時計の縁 (外側)	ライムグリーン	ピンク

4 プログラムの説明

3 章で述べたアプリケーションを作成するためには次に示す機能を実装しなければならない. 本章ではこれらを実装するプログラムの説明を行う. プログラムリスト全体については付録!を参照してほしい.

- ヘッダファイルの記述, オブジェクト形式マクロの宣言
- 初期化設定
- ウィンドウのリサイズへの対応
- タイマーを用いた時間の更新
- マウス入力の制御
- アナログ時計の針を表示する機能
- アナログ時計のインデックス, 文字盤を表示する機能
- 年, 月, 日, 曜日, 時, 分, 秒の文字列を表示する機能
- 時計の縁が回転する機能

4.1 ヘッダファイルの記述, オブジェクト形式マクロの宣言

プログラムの実行に必要なヘッダファイルの記述, および必要なマクロを定義する. リスト 1 にヘッダファイルの記述, および必要なマクロを定義したコードを示す.glut.h が c 言語で OpenGL を扱うためのライブラリである. また,time.h は時間の取得を行うためのライブラリである.

本アプリではウィンドウサイズは 320×320 に固定する. このため, リスト 1 の 8 行目および 9 行目ではウィンドウのサイズを定義している.

リスト 11 行目から 13 行目行目ではグローバル変数の定義を行っている. 変数 dispMode はライトモード,

ダークモードの管理を行うための変数である.dispMode が 1 のときライトモード,0 のときダークモードである.dispMode のデフォルト値はライトモードなっている. 変数 loop1,loop2 はアナログ時計の縁を回転させるための変数である.

リスト 1: 定数および変数の定義

```
1  #include<GL/glut.h>
2  #include<stdio.h>
3  #include<time.h>
4  #include<math.h>
5  #include<string.h>
6
7  // windowのサイズを定義
8  #define WINDOW_W 320
9  #define WINDOW_H 320
10
11 int dispMode =0; // 0 : LIGHTMODE 1 : DARKMODE
12 double loop1=0; //use for design rotation
13 double loop2=0; //use for design rotation
```

4.2 初期化設定

メイン関数では全体の初期化および設定を行う. リスト 2 にメイン関数のコードを示す.

リスト 2: main 関数

```
1  int main(int argc,char **argv){
2  // 初期化処理
3      // 引数処理
4      glutInit(&argc,argv);
5      // 初期Windowサイズ設定
6      glutInitWindowSize(WINDOW_W,WINDOW_H);
7      // 新規Window作成
8      glutCreateWindow("clock");
9      // 関数登録
10     glutDisplayFunc(Display);
11     glutReshapeFunc(Reshape);
12     glutMouseFunc(Mouse);
13     glutTimerFunc(500,Timer,0);
14     // display初期化
15     glutInitDisplayMode(GLUT_RGBA);
16     glClearColor(0.96,0.96,1.0,1.0);
17     // メインループ
18     glutMainLoop();
19     return 0;
20 }
```

メイン関数の処理の説明を次に示す.

1. 引数の処理を glutInit 関数で行う. 本アプリでは引数は使用しないから, ここでは glutInit 関数に形式的に引数を渡しているだけである (リスト 2 の 4 行目).

2. 最初に開くウィンドウのサイズを指定する. ウィンドウのサイズの初期設定は glutInitWindowSize 関数で行う.glutInitWindowSize 関数の引数は (横幅, 縦幅) である. ここでは, オブジェクト形式マクロで定義した WINDOW_W を横幅,WINDOW_H を縦幅とする (リスト 2 の 6 行目).

3. 開くウィンドウのサイズが決まったから, ウィンドウを生成する. ウィンドウの生成は glutCreateWindow 関数で行う.glutCreateWindow 関数の引数として渡している文字列は図 1 において, ウィンドウの左上に表示されている文字列である. リスト 2 の 8 行目では「clock」という文字列を glutCreateWindow 関数に渡しているから図 1 のウィンドウの左上には「clock」と表示されている.

4. リスト 2 の 10 行目から 13 行目はイベントによって呼び出される関数を定義している. 本アプリにおいて, イベント (例としてユーザーからの入力やタイマーによる時間経過) はループを用いて逐一見張っ

ている仕組みではない. 本アプリのイベントに対する仕組みは, 普段は何もせず, イベントが起こったときにそれに応じた処理を行うものである. このような仕組みをイベント駆動型プログラミングという. また, 呼び出される関数をコールバック関数という. リスト 2 では 4 つイベントに対してコールバック関数を設定している. それぞれのコールバックの呼び出されるイベントと処理内容の概要を次に示す.

- glutDisplayFunc ... ウィンドウの表示内容を更新する Display 関数を呼び出す.
- glutReshapeFunc ... ウィンドウのサイズが変更されたときに, 座標系およびウィンドウのサイズに関する設定する Reshape 関数を呼び出す.
- glutMouseFunc ... マウスの移動やクリックが発生したときに, マウスの移動やクリックに対する処理を行う Mouse 関数を呼び出す.
- glutTimerFunc ... 第一引数の時間 (ミリ秒) が経過したときに, タイマーの処理を行う Timer 関数を呼び出す.

5. リスト 2 の 15 行目から 16 行目では生成したウィンドウに初期の背景を描画する処理を行っている.glutInitDisplayMode 関数は色の指定をどのように行うかを設定している. ここでは色を RGBA, つまり赤 (Red), 緑 (Green), 青 (Blue), 透明度 (Alpha) の 4 つの変数で指定する設定を行っている. そして glClearColor 関数で初期の背景を描画する処理を行っている. 注意として,glClearColor 関数の引数は 0 から 1 までの値で色を指定する. ここでは, すべての値がほぼ 1 であるため, 白に近い色が描画される.

6. 初期設定が終了したから, メインのループに入る関数である glutMainLoop 関数を実行する.

4.3 ウィンドウのリサイズへの対応

ウィンドウのリサイズへの対応について, 仕様とプログラムでの実装部分を説明する. 本アプリではウィンドウのリサイズを行っても, 元のサイズ (320×320) に戻される仕様になっている. これには 2 つの理由がある.1 つ目は, アナログ時計を描画する部分において, ウィンドウのサイズに応じてアナログ時計の大きさを変化させるときに, 中心からの距離の指定が複雑になることを防ぐためである. 2 つ目は, 文字列を表示している関数は自由に拡大縮小ができないため, 画面の大きさに対して不格好な描画になってしまうためである.

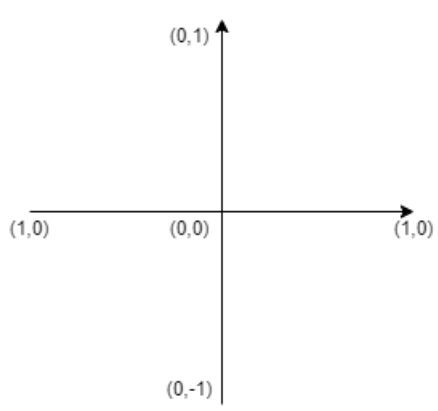
ウィンドウのサイズ変更イベントの処理を行う関数は Reshape 関数であった. リスト 3 に Reshape 関数のコードを示す. リスト 3 において,Reshape 関数の引数は新しいウィンドウの幅, 高さ h である.

リスト 3: Reshape 関数

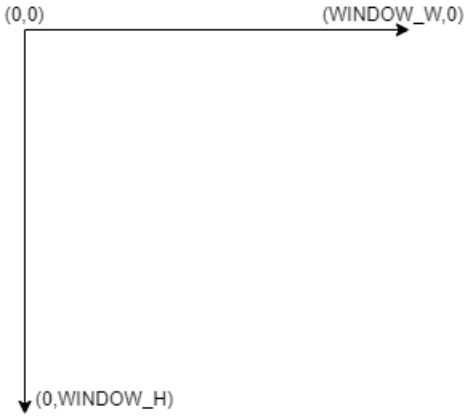
```
1  void Reshape(int w,int h){
2      //printf("ウィンドウの幅と高さ=%d x %d\n",w,h);
3      glViewport(0,0,w,h);
4      glMatrixMode(GL_MODELVIEW);
5      glLoadIdentity();
6      gluOrtho2D(0,w,0,h);
7      glScaled(1,-1,1);
8      glTranslated(0,-h,0);
9
10     //windowサイズ固定
11     glutReshapeWindow(WINDOW_W,WINDOW_H);
12 }
```

Reshape 関数の処理内容について説明する. ウィンドウの座標系はウィンドウを更新するたびに初期設定に戻ってしまう. ウィンドウの初期設定における座標系を図 2a に示す. 図 2a の座標系では描画する図形の頂点の値を実数で与えないといけないため, 扱いにくい. そこで図 2b に示す座標系に設定しなおす. リスト 3 の 2 行目から 8 行目では, 図 2a の座標系を, リスト 2b の座標系にする設定を行っている.

ウィンドウサイズの固定はリスト 3 の 11 行目で行っている.glutReshapeWindow 関数にウィンドウの幅, 高さを引数として渡すことで内部でウィンドウのリサイズを行うことができる.



(a) 初期設定の座標系



(b) 変換後の座標系

図 2: 座標系の設定

4.4 タイマーを用いた画面描画の更新

タイマーによって一定時間おきに画面描画を更新する仕組みについて説明する. 実装したいことはタイマーによって一定おきにイベントが発生することと, 画面描画を更新することに切り分けられる. 本節では, 前者のタイマーによって一定おきにイベントが発生する仕組みについて説明する. 後者はここでは, 画面描画を更新する Display 関数がうまくやってくれると考える.

第一引数の時間 (ミリ秒) が経過したときに, タイマーの処理を行う関数は Timer 関数であった.Timer 関数のコードをリスト 4 に示す.Timer 関数はメイン関数 (リスト 2) の 13 行目に呼ばれる. ここでは 500 ミリ秒経過後にリスト 4 の Timer 関数が呼ばれる設定になっている.

Timer 関数の処理内容は, まず 2 行目でディスプレイの表示内容を更新するためのイベントを発生させる glutPostRedisplay 関数が実行される. これによって display 関数が呼び出されて画面表示が更新される. 次に,glutTimerFunc を再度実行する (リスト 4 の 3 行目).glutTimerFunc 関数に登録されたタイマーは 1 度しか実行されないため, 反復的にタイマーを利用するためにはイベントの設定を再度行う必要がある.

リスト 4: Timer 関数

```
1 void Timer(int value){
2     glutPostRedisplay();
3     glutTimerFunc(500,Timer,0);
4 }
```

4.5 マウス入力の制御

マウスの左クリックによってライトモードとダークモードが切り替わる仕組みについて説明する. マウスの移動やクリックが発生したときに, その動作に対する処理を行う関数は Mouse 関数であった. Mouse 関数のコードをリスト 5 に示す.Mouse 関数の引数は, クリックされたボタンを示す b, ボタンの状態 (押されたのか, 離されたのか) を示す s, ボタンの座標 (x,y) の 4 つである.

リスト 5: Mouse 関数

```
1 void Mouse(int b,int s,int x,int y){
2     if(b==GLUT_LEFT_BUTTON){
3         if(s==GLUT_UP){
4             if(dispMode==1){
5                 dispMode=0;
6             }else{
```

```
7                   dispMode=1;
8               }
9           }
10          if(dispMode){
11              glClearColor(0.15,0.15,0.15,1.0);
12          }else{
13              glClearColor(0.96,0.96,1.0,1.0);
14          }
15      }
16  }
```

Mouse 関数の処理内容について説明する. リスト 5 の 2 行目から 9 行目では左クリックがされたときに dispMode を切り替える処理を行っている. リスト 5 の 2 行目の if 文で, イベントが起きたボタンが「左ボタン」であることを判定し,3 行目の if 文でボタンが「離された」ことを判定している. このように, マウスイベントの判定は,「左ボタンがクリックされたか」という事象を, イベントがあったボタンの種類, 押されたのか離されたのか, という 2 つに分解して判定している. そして,5 行目から 8 行目で 2 つのモードの切り替えを行っている.

リスト 5 の 10 行目から 14 行目では dispMode の変更に伴って, 背景色を変更したいから,glClearColor 関数を実行する処理を行っている.

4.6 アナログ時計の針を表示する機能

画面表示を行う関数として Display 関数を作成し, 次の機能を実装する.Display 関数の設計として, 画面に表示する順番がある. アナログ時計の描画において, 最も最前面に描画するべき情報は時計の針である. 次に, 時計のインデックス, 文字盤, 年月日に関する文字列の 3 つが同程度に重要である. 時計の縁はデザイン性を高めるためのものであるから最も重要度が低い. したがって, フロー処理においてこれらを描画する順番は箇条書きの順番と一致する.Display 関数における 4 つの機能の実装プログラムの説明は, プログラムの必要な部分を抜粋して行う. このため描画する順番がわかりずらい. プログラムソース全体は付録!に載っているから, 描画の順番は付録!の!を参照してほしい.

1. 時計の縁が回転する機能
2. 年, 月, 日, 曜日, 時, 分, 秒の文字列を表示する機能
3. アナログ時計のインデックス, 文字盤を表示する機能
4. 時計の針を描画する機能

アナログ時計の針を表示する機能について解説する. アナログ時計の「針を表示する機能」を実装するコードを抜粋したものをリスト 6 に示す. リスト 6 には Display 関数,calPosition 関数,drawLine 関数の 3 つの関数がある.

リスト 6: 針の描画の実装

```
1   void Display(void){
2       int i; //ループ用
3       char *timestr; // 時間情報表示用文字列
4       // 画面サイズ取得
5       int xc = glutGet(GLUT_WINDOW_WIDTH)/2;
6       int yc = glutGet(GLUT_WINDOW_HEIGHT)/2+30; // y軸方向の中心は30ずらす.
7
8       // 針の角度
9       double thetas,thetam,thetah;
10      // 針の座標
11      int xs,ys,xm,ym,xh,yh;
12      // 針の長さ
13      int ls=80;
14      int lm = 105;
15      int lh = 90;
16
```

```
17      // 描画クリア
18      glClear(GL_COLOR_BUFFER_BIT);
19
20  // 時間取得
21      time_t tt;
22      struct tm *ts;
23      time(&tt);
24      ts = localtime(&tt);
25
26  // 針の角度,座標を計算
27      thetas = 2*M_PI*ts->tm_sec/60;
28      thetam = 2*M_PI*(60*ts->tm_min+ts->tm_sec)/3600;
29      thetah = 2*M_PI*(3600*(ts->tm_hour%12)+60*ts->tm_min+ts->tm_sec)/43200;
30      calPosition(&xs,&ys,xc,yc,ls,thetas);
31      calPosition(&xm,&ym,xc,yc,lm,thetam);
32      calPosition(&xh,&yh,xc,yc,lh,thetah);
33
34  // 針を描画
35      //時針描画
36      if(dispMode){
37      glColor3ub(255,255,255);
38      }else{
39      glColor3ub(0,0,0);
40      }
41      glLineWidth(5.0);
42      drawLine(xc,yc,xh,yh);
43      //分針描画
44      glLineWidth(3.0);
45      drawLine(xc,yc,xm,ym);
46      //秒針描画
47      glLineWidth(2.0);
48      glColor3ub(255,0,0);
49      drawLine(xc,yc,xs,ys);
50
51      glFlush();
52  }
53
54  // 極座標と直交座標を変換
55  //極座標(r,theta)を(xc,yc)を原点とした直交座標(x,y)に変換
56  void calPosition(int *x,int *y,int xc,int yc,int r,double theta){
57      *x = xc+r*sin(theta);
58      *y = yc-r*cos(theta);
59  }
60
61  // lineを描画
62  //(x1,y1)と(x2,y2)を結ぶ直線を描画
63  void drawLine(int x1,int y1,int x2,int y2){
64      glBegin(GL_LINES);
65      glVertex2i(x1,y1);
66      glVertex2i(x2,y2);
67      glEnd();
68  };
```

5　ビルド方法の説明

6　実行結果とその説明

!付録

参考文献

[1] 国立高専機構長野高専,http://www.nagano-nct.ac.jp/ , 閲覧日 2020 年 8 月 5 日