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ことを目的とする.

2　実行環境

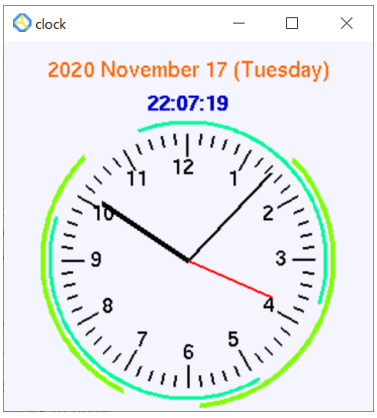
実行環境を 1 に示す.gcc とは「GNU Compiler Collection」の略称で,GNU プロジェクトが公開しているコンパイラのことである. make は Makefile にプログラムのコンパイルやリンクの方法を指示することで, コンパイルを簡単に行うことができるツールのことである. make を用いることは,gcc コンパイル時に, 長いオプションを入力しなくてよい, ファイルの更新を取得して必要なものだけをコンパイルしてくれるという利点がある.

表 1: 実行環境

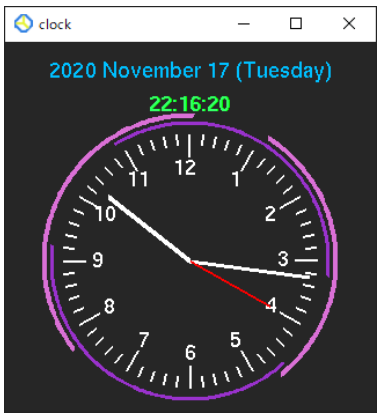
CPU	Intel(R) Core(TM) i7-6500U 2.50GHz
メモリ	16.0GB DDR4
OS	Microsoft Windows 10 Home
gcc	version 9.3.0
make	version 4.3

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.

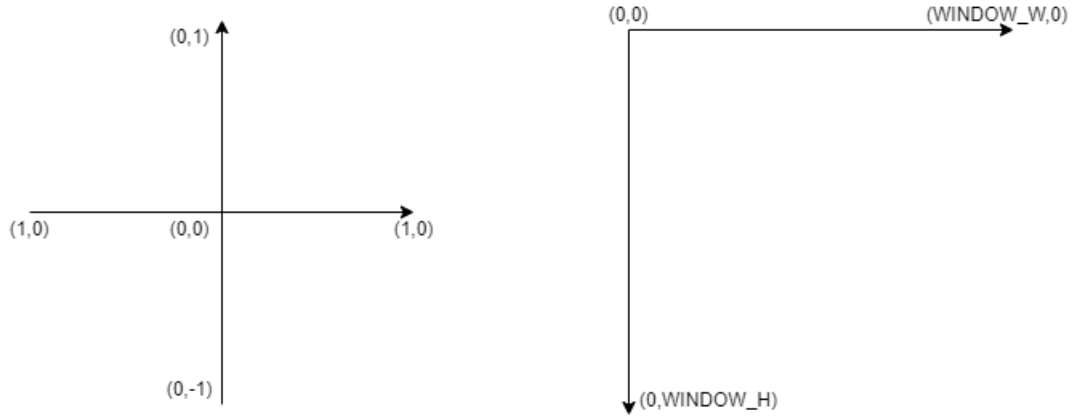
リスト 3: Reshape 関数

```
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

```
void Display(void){
    int i; //ループ用
    char *timestr; // 時間情報表示用文字列
    // 画面サイズ取得
    int xc = glutGet(GLUT_WINDOW_WIDTH)/2;
    int yc = glutGet(GLUT_WINDOW_HEIGHT)/2+30; // y軸方向の中心は30ずらす.

    // 針の角度
    double thetas,thetam,thetah;
    // 針の座標
    int xs,ys,xm,ym,xh,yh;
    // 針の長さ
    int ls=80;
    int lm = 105;
    int lh = 90;

    // 描画クリア
    glClear(GL_COLOR_BUFFER_BIT);

// 時間取得
    time_t tt;
    struct tm *ts;
    time(&tt);
    ts = localtime(&tt);

// 針の角度,座標を計算
    thetas = 2*M_PI*ts->tm_sec/60;
    thetam = 2*M_PI*(60*ts->tm_min+ts->tm_sec)/3600;
    thetah = 2*M_PI*(3600*(ts->tm_hour%12)+60*ts->tm_min+ts->tm_sec)/43200;
    calPosition(&xs,&ys,xc,yc,ls,thetas);
    calPosition(&xm,&ym,xc,yc,lm,thetam);
    calPosition(&xh,&yh,xc,yc,lh,thetah);

// 針を描画
    //時針描画
    if(dispMode){
    glColor3ub(255,255,255);
    }else{
    glColor3ub(0,0,0);
    }
    glLineWidth(5.0);
    drawLine(xc,yc,xh,yh);
    //分針描画
    glLineWidth(3.0);
    drawLine(xc,yc,xm,ym);
    //秒針描画
    glLineWidth(2.0);
    glColor3ub(255,0,0);
    drawLine(xc,yc,xs,ys);

    glFlush();
}

// 極座標と直交座標を変換
//極座標(r,theta)を(xc,yc)を原点とした直交座標(x,y)に変換
void calPosition(int *x,int *y,int xc,int yc,int r,double theta){
    *x = xc+r*sin(theta);
    *y = yc-r*cos(theta);
}

// 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 };
```

Display 関数の処理の内容を説明する．リスト 6 の 5 行目および 6 行目では, ウィンドウの中央の座標を取得して変数 xc,xy に代入している．glutGet 関数はウィンドウや画面の情報を取得するための関数で, 引数として「GLUT_WINDOW_WIDTH」を与えるとウィンドウの幅の情報が戻り値として得られる. 同様に引数として「GLUT_WINDOW_HEIGHT」を与えることで, ウィンドウの高さの情報が取得できる. そして取得した値を 2 で割ることで, 画面の中央の座標を得ることができる.6 行目で yc の値に 30 を足しているのは, 画面中央を基準に時計を描画すると, 文字列を表示する場所が狭くなってしまうためである.yc に 30 を足すことで, 時計の中心を y 軸方向 30pixel 下に移動している.

リスト 6 の 9 行目から 15 行目では秒針, 分針, 時針の 3 つの描画位置を計算するための変数を定義している.

リスト 6 の 18 行目では, 画面表示をクリアな状態にする glClear 関数を実行している. 引数として与えている「GL_COLOR_BUFFER_BIT」はカラーバッファとよばれる色情報を格納するメモリのことで, これを塗りつぶすことで, 画面表示をクリアしている.

時計を作成するために, 時間の取得を行う. 時間の取得を行っている部分はリスト 6 の 21 行目から 24 行目である.「time.h」の関数を用いるため, リスト 1 の 3 行目で「time.h」を読み込んでいる. 時間の取得は time_t 型の変数を time 関数にわたすことで取得することができる.

我々がほしい情報は現在時刻は tm 構造体が保持しているため,time_t 型から変換を行う必要がある. この変換を行っているのが, リスト 6 の 24 行目である.tm 構造体の構造はリスト 7 のようになっている．現在時間の情報の取得方法の例として, 日付の情報がほしい場合は「ts->tm_mday」と記述することで取得できる. 他の情報の取得方法もリスト 7 を参考に取得できる.

リスト 7: tm 構造体

```
1  struct tm{
2    int tm_sec; // 秒(0~60). 60は閏秒対応のため
3    int tm_min; // 分(0~59).
4    int tm_hour; // 時間(0~23).
5    int tm_mday; // 日(1~31).
6    int tm_mon; // 1月からの通算月数(0~11).
7    int tm_year; // 1900年からの通算年数.
8    int tm_wday; // 曜日(0~6).0が日曜日で6が土曜日.
9    int tm_yday; // 1月1日からの通算日数(0~365).
10   int tm_isdst; // 夏時間が有効かどうかのフラグ.
11 }
```

時間の取得ができたから, アナログ時計の針の角度および座標を計算する. リスト 6 では 27 行目から 32 行目である．27 行目から 29 行目ではそれぞれの針の角度を計算している. 角度は図 3 に示すように時計の 12 時の位置を 0 度としている.

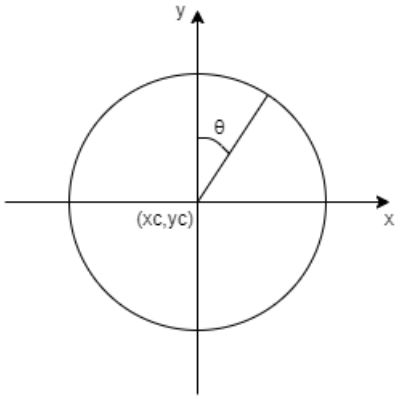


図 3: 時計の針の角度

時刻 h 時 m 分 s 秒のときの秒針の角度 θ_s は式 (1) で与えられる. 同様に分針の角度 θ_m は式 (2), 分針の角度 θ_h は式 (3) で計算できる.

$$\theta_s = \frac{2\pi s}{60} \tag{1}$$

$$\theta_m = \frac{2\pi(60m+s)}{3600} \tag{2}$$

$$\theta_h = \frac{2\pi(2600h+60m+s)}{43200} \tag{3}$$

角度が計算できたから, 中心からの距離と角度の情報をもとに, 針先の座標を計算する. 針先の座標の計算はリスト 6 の 30 行目から 32 行目に示すように calPosition 関数で行っている.calPosition 関数の定義はリスト 6 の 56 行目から 59 行目にある.calPosition 関数は, 針先の座標を格納する変数 x,y のポインタ, 中心座標 (xc,yc), 中心からの距離 r, 角度 theta の 6 つを引数としている. 計算結果は戻り値ではなく x,y に代入される. 極座標 (r,θ) から直交座標 (x,y) の変換は式 (4) および式 (5) でできる.

$$x = x_c + l\sin\theta \tag{4}$$

$$y = y_c - l\cos\theta \tag{5}$$

針の座標が計算できたから, 描画を行う. プログラムではリスト 6 の 36 行目から 49 行目である.36 行目から 40 行目では, 針の色の設定を行っている. まず, 表示モードで色の場合分けがあるため,if 文で dispMode でどちらのモードかを判定し, 処理を分岐させる.

そして,glColor3ub 関数で色の設定で行う.glColor3ub 関数は引数として (R,G,B) の情報を 0 から 255 の整数で与えることで色の設定が行われる. 設定は再度 glColor3ub 関数で違う色に設定するまで有効である. ダークモード (dispMode=1) のとき, 針の色を (255,255,255), つまり白に設定している. ライトモード (dispMode=0) のときは, 針の色を (0,0,0), つまり黒に設定している. このため, 図 1 に示したように, 時針と分針がライトモードのときは黒, ダークモードのときは白で表示される. 秒針が赤で描画されているのは,48 行目で glColor3ub 関数を呼び出して, 色を (255,0,0), つまり赤に設定してから秒針を描画しているからである.

色の設定ができたから, 線の太さの設定を行う. プログラムでは 41,44,47 行目である.41 行目では, 時針の太さを glLineWidth 関数で設定している. glLineWidth 関数は引数に太さを数値で与えると, 値に応じた太さで線が描画させる. 同様に,44 行目で分針,47 行目で秒針の太さを設定している. 図 1 を見ると, それぞれの針で太さが変わっていることがわかる.

時針の描画は 42 行目, 分針の描画は 44 行目, 秒針の描画は 49 行目で行っている. 針の描画を行う関数は drawLine 関数である.drawLine 関数はリスト 1 の 63 行目から 68 行目で定義している.drawLine 関数は引数として 2 点の座標 (x1,y1),(x2,y2) を与えると, この 2 点を結ぶ線を描画する機能をもつ. 線を結ぶ方法は 64 行目の glBegin 関数で画面描画を行うことを宣言する.67 行目の glEnd 関数は glBegin 関数と対になる関数である. 直線を描くための点は glVertex2i 関数で指定する.65 および 66 行目のように,glVertex2i 関数に座標をわたすことで直線を描くための点を指定できる. そして,glBegin 関数に「GL_LINES」を引数として指定することで直線を描くことができる.

ここまでで描画した内容はキューにためられているだけで, まだ描画されていない. このキューの内容を実行して画面に反映させる関数がリスト 1 の 51 行目の glFlush 関数である.

4.7 アナログ時計のインデックス, 文字盤を表示する機能

アナログ時計のインデックス (目盛り) および文字盤を表示する機能の実装について説明する. リスト 8 にインデックスおよび文字盤を表示するコードを示す.

リスト 8: インデックスと文字盤の実装

```
1  void Display(void){
2    // インデックス描画用
3    double l,theta;
4    int x1,x2,y1,y2;
5    char s[3];
6
7    // ---中略---
8
9    // インデックス描画
10   for(i=1;i<=60;i++){
11       if(dispMode){
12         glColor3ub(255,255,255);
13       }else{
14         glColor3ub(0,0,0);
15       }
16       glLineWidth(2.0);
17       l=100; // インデックスの先端を長さ100にする
18       if(i%5==0){  // 5の倍数の針は長くする
19       l = 90; // インデックスの終端を長さ90にする
20       }
21       theta = 2*M_PI*i/60;
22       calPosition(&x1,&y1,xc,yc,l,theta);
23       l = 110;
24       calPosition(&x2,&y2,xc,yc,l,theta);
25       drawLine(x1,y1,x2,y2);
26
27       if(i%5==0){ // 5の倍数のとき文字を表示
28           sprintf(s,"%d",i/5);
29           l =80; // 文字表示位置を80にする
30           calPosition(&x2,&y2,xc,yc,l,theta);
31           if(i/5<10){ // 一桁表示用
32               glRasterPos2i(x2-5,y2+5);
33               glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,s[0]);
34
35           }else{ // 二桁表示用
36               glRasterPos2i(x2-14,y2+5);
37               glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,s[0]);
38               glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,s[1]);
39           }
40       }
41   }
42 }
```

リスト 8 の 2 行目から 5 行目では, 実装に必要な変数の定義を行っている. そして, リスト 8 の 10 行目から 41 行目の for 文がインデックスと文字盤を描画している部分である.for 文の処理の大まかな内容は, インデックスの表示を for 文を用いて, 変数 i が 1 から 60 までの 60 回ループすることで行っている. 文字盤の表示は 60 回ループするなかで,i が 5 の倍数のときに行っている.
リスト 8 の 11 行目から 15 行目では色の設定を行っている. そしてリスト 8 の 16 行目ではインデックスを描画する太さを 2.0 に設定している.

リスト 8 の 17 行目から 25 行目がインデックスの先端および終端の座標を計算して, 描画する部分である. 時計の針の描画は, 画面の中心と先端を線で結ぶことで行っていた. インデックスの描画は, 先端 (中心に近いほう) の座標 (x1,y1) と終端の座標 (x2,y2) の二つを計算する必要がある. 先端と終端の角度は同じであるから長さを変えて計算を行えばよい. リスト 8 の 17 行目から 22 行目では時計の先端の座標を計算している. 長さはインデックスが 5 の倍数のとき, つまり文字盤を表示するとき 90 に設定し, それ以外の場合は 100 にしている. こうすることで, 図 1 に示すように, 文字盤が表示されているインデックスの長さを伸ばして, 見やすくしている. 長さの設定ができたから, リスト 8 の 22 行目に示すように,calPosition 関数 (リスト 6 の 56 行目から 59 行目) を用いて先端の座標を計算する. 同様に, リスト 8 の 23,24 行目で, 長さを 110 に設定して終端の座標を計算している. 最後に drawLine 関数に先端と終端の座標を引数として与えることでインデックスが描画される.

リスト 8 の 27 行目から 40 行目ではループ変数 i が 5 の倍数のときに文字盤を表示する機能を実装している. まず,27 行目の if 文で変数 i が 5 の倍数かどうかを判定し, 処理が分岐する. 変数 i が 5 の倍数のとき 28 行目の sprintf 関数を用いて文字列 s に「i/5」の文字列を代入する. 例えば変数 i の値が 5 のとき s には「1」の文字列が入る. sprintf は printf で標準出力される文字列が第一引数の変数に代入される. このため,「i/5」の値が文字列 s に代入される. そして,29 行目および 30 行目で文字列を描画する位置を計算する.

文字盤の表示は文字列が一桁 (1〜9) のときと二桁 (10〜12) のときで分けて処理を行っている. リスト 8 の 31 行目から 33 行目が文字列が一桁のときの処理である. まず, 文字列を描画する座標を glRasterPos2i 関数に与える. calPosition 関数の計算結果から x 座標を −5,y 座標を +5 しているのは表示位置を微調整しているためである. そして,glutBitmapCharacter 関数で文字列の描画を行う.glutBitmapCharacter 関数の第一引数は描画するフォントの種類を指定している. 第二引数は表示する文字を与える.glutBitmapCharacter 関数は一文字ずつしか描画できないが, 表示文字が一文字のため,s[0] を与えている.

二桁 (10〜12) を表示する処理はリスト 8 の 36 行目から 38 行目である. まず, 一桁目の描画と同様に glRasterPos2i で描画する座標を指定する. ここでも表示位置の微調整を行っている. そして glutBitmapCharacter 関数で 1 文字目を表示する.glutBitmapCharacter 関数には自動で次の文字の描画位置に座標をずらす機能があるから, 続けて 2 文字目を表示する. これで 2 桁のときも画面に描画できる. リスト 8 の 5 行目で文字列 s を要素数 3 で定義しており,3 文字目を描画しなくてよいのかと疑問に思うかもしれないが,3 文字目はナル文字と呼ばれる文字列の終わりを示す文字が入っているため描画する必要はない.

4.8 年, 月, 日, 曜日, 時, 分, 秒の文字列を表示する機能

日付の情報を画面に描画する機能の実装について説明する. リスト 9 に日付情報を描画する機能を実装するためのコードを示す.

リスト 9: 日付情報を描画する機能の実装

```
1  void Display(void){
2    char *timestr; // 時間情報表示用文字列
3    int month,wday; // 月の番号,曜日の番号取得用
4    int timestr_len = 12; //year+space+space+month+space+(+)+\0
5    // 月の名前を定義
6    char month_eg[12][10] = {{"January"},{"February"},{"March"},{"April"},{"May"}
7    ,{"June"},{"July"},{"August"},{"Septermber"},{"October"},{"November"},{"December"}};
```

```
8     // 曜日の名前を定義
9     char wday_eg[7][10] = {{"Sunday"},{"Monday"},{"Tuesday"},{"Wednesday"},{"Thursday"}
10    ,{"Friday"},{"Saturday"}};
11
12    -----中略-----
13
14    // year,month,dayを表示
15    month = ts->tm_mon; // 月の番号を取得
16    wday =  ts->tm_wday; // 曜日の番号を取得
17    // 可変文字列生成
18    timestr_len+=strlen(month_eg[month])+strlen(wday_eg[wday]); // 文字数計算
19    timestr = (char *)malloc(timestr_len*sizeof(char)); // 配列確保
20    // timestrに文字列書き込み
21    sprintf(timestr,"%d %s %02d (%s)",1900+ts->tm_year,month_eg[month],ts->tm_mday
22    ,wday_eg[wday]);
23    // 文字列表示
24    if(dispMode){
25    glColor3ub(0,191,255);
26    }else{
27    glColor3ub(255,102,0);
28    }
29    Printstr(xc-(18*timestr_len/4),30+1,timestr,timestr_len); // 文字列表示
30    // 領域解放
31    free(timestr);
32
33  //hour,min,secを表示
34    // 文字列生成
35    timestr = (char *)malloc(9*sizeof(char)); // 配列確保
36    // timestrに文字列書き込み
37    sprintf(timestr,"%02d:%02d:%02d",ts->tm_hour,ts->tm_min,ts->tm_sec);
38    timestr_len = strlen(timestr); // 文字列の長さを取得
39    // 文字列表示
40    if(dispMode){
41    glColor3ub(0,255,0);
42    }else{
43    glColor3ub(0,0,205);
44    }
45    // 文字列表示(太文字用)
46    Printstr(xc-(18*timestr_len/4)+1,60,timestr,timestr_len);
47    if(dispMode){
48    glColor3ub(51,255,102);
49    }else{
50    glColor3ub(0,0,205);
51    }
52    Printstr(xc-(18*timestr_len/4),60,timestr,timestr_len); // 文字列表示
53    // 領域解放
54    free(timestr);
55  }
56
57  // 文字列描画
58  //座標(x,y)を初期位置として文字列str(文字列の長さstrlen)を画面に描画
59  void Printstr(int x,int y,char *str,int strlen){
60      int i;
61      glRasterPos2i(x,y);
62      for(i=0;i<strlen;i++){
63          glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,str[i]);
64      }
65  }
```

リスト 9 の 2 行目から 10 行目では日付情報の描画に必要な変数を定義している.4 行目の変数 timestr_len は文字列を格納する配列を動的生成するために文字数を計算している. ここでは月名および曜日名のように文字列が可変である部分を除いた文字数を計算している. 画面に描画したい文字は「2020␣[月名]␣06␣([曜日])」のような形である.([ ] で囲んでいる部分が可変).「␣」は空白 (スペース) を表している. 年は 4 桁を仮定し, 日は 2 桁で表示する仕様にすると, 固定で必要な文字は 11 文字である. 変数 timestr_len には 11 文字にナル文字用の 1 文字を足した 12 文字を代入している.

6 行目の month_eg 配列は月の名前を保持するための配列である. また,9 行目の wday_eg 配列は曜日の名

前を保持するための配列である.

リスト 9 の 15 行目から 31 行目では図 1 の「2020 November 17 (Tuesday)」にあたる部分を描画している. 15 行目では月の番号,16 行目では曜日の番号を取得している.18 行目から 22 行目では画面に表示する文字列の生成を行っている. 18 行目では strlen 関数を用いて, 表示すべき月および曜日の文字数を計算し, 変数 timestr_len に加算している.18 行目で必要な文字列の長さがわかったから, 領域の確保を行う. 領域の確保は 19 行目で malloc 関数を用いて行っている. 領域が確保できたから文字列 timestr に文字列を代入する. 文字列の代入は 21 行目で sprintf 関数を用いて行っている. 日の文字数を 2 桁にする処理も sprinf 関数で行っている.

表示する文字列が準備できたから, 描画を行う. 描画はリスト 9 の 24 行目から 29 行目で行っている. まず 24 行目から 27 行目で描画する色の設定を行う. ダークモードのときはライトブルー, ライトモードのときはオレンジになるように設定している.29 行目で文字列の描画を Printstr 関数を用いて行っている.Printstr 関数の定義はリスト 9 の 59 行目から 65 行目にある.Printstr 関数は引数として 1 文字目を描画する座標 (x,y), および文字列, および文字列の長さを与える必要がある.Printstr 関数の内部では, 引数として受け取った文字列を glutBitmapCharacter で 1 文字ずつ表示する処理を行っている. これによって文字列が画面に表示される. 最後に malloc で確保した領域を解放する. 領域の解放は 31 行目で free 関数を用いて行っている.

同じ方法で時, 分, 秒の文字列を描画する. リスト 9 において時, 分, 秒の文字列を描画している部分は 35 行目から 54 行目である. まず, 配列の確保を動的に行う. 表示したい文字列は「22:16:20」のような形をしている. すべての桁を 2 文字で表示する仕様にすると, 必要な文字数は 8 文字である. リスト 9 の 35 行目では,8 文字にナル文字の 1 文字を足した 9 文字の長さの配列を確保している. そのほかの処理は日付情報を画面に描画したときと同様である.46 行目で文字列を再度描画しているのは, 表示する位置を y 方向に 1px ずらして, 表示を太文字にするためである. このため, 図 1 で日付の表示と, 太字で描画した時間の描画では, 文字の太さに違いが見られる.

4.9 時計の縁が回転する機能

時計の縁を描画し, 回転させる機能の実装について説明する. リスト 10 に時計の縁を描画し, 回転させる機能を実装するためのコードを示す.

リスト 10: 時計の縁の実装

```
1  void Display(void){
2
3    -----中略-----
4
5    // デザイン描画
6    l=128;
7    if(dispMode){
8    glColor3ub(218,112,214);
9    }else{
10   glColor3ub(127,255,0);
11   }
12   drawDesign(xc,yc,0,130,l,loop1);
13   l=124;
14   if(dispMode){
15   glColor3ub(38,38,38);
16   }else{
17   glColor3ub(245,245,255);
18   }
19   drawDesign(xc,yc,-1,131,l,loop1);
20
21
22
23   l=128;
24   if(dispMode){
25   glColor3ub(218,112,214);
```

```
26    }else{
27    glColor3ub(127,255,0);
28    }
29    drawDesign(xc,yc,160,270,l,loop1);
30    l=124;
31    if(dispMode){
32    glColor3ub(38,38,38);
33    }else{
34    glColor3ub(245,245,255);
35    }
36    drawDesign(xc,yc,159,271,l,loop1);
37
38
39
40
41
42    l=121;
43    if(dispMode){
44    glColor3ub(153,50,204);
45    }else{
46    glColor3ub(0,250,154);
47    }
48    drawDesign(xc,yc,70,200,l,loop2);
49    l=118;
50    if(dispMode){
51    glColor3ub(38,38,38);
52    }else{
53    glColor3ub(245,245,255);
54    }
55    drawDesign(xc,yc,69,201,l,loop2);
56
57
58
59    l=121;
60    if(dispMode){
61    glColor3ub(153,50,204);
62    }else{
63    glColor3ub(0,250,154);
64    }
65    drawDesign(xc,yc,240,380,l,loop2);
66    l=118;
67    if(dispMode){
68    glColor3ub(38,38,38);
69    }else{
70    glColor3ub(245,245,255);
71    }
72    drawDesign(xc,yc,239,381,l,loop2);
73
74  // デザイン回転
75    loop1+=0.05;
76    loop2-=0.1;
77    if(loop1>=2*M_PI){
78        loop1=0;
79    }
80    if(loop2<=-2*M_PI){
81        loop2=0;
82    }
83  }
84
85  // デザインを描画
86  // (xc,yc)を原点とする極座標をとる.
87  // この極座標でthetaStartからtheraEndまで,長さlの扇形を描画する.
88  // loopは扇形を回転させる変数.
89  void drawDesign(int xc,int yc,int thetaStart,int thetaEnd,int l,double loop){
90      int x,y,i;
91      double theta;
92      glBegin(GL_POLYGON);
93      for(i=thetaStart;i<=thetaEnd;i++){
94          theta= 2*M_PI*i/360;
95          calPosition(&x,&y,xc,yc,l,theta+loop);
```

```
96          glVertex2i(x,y);
97      }
98      glEnd();
99  }
```

リスト 10 の Display 関数において, 時計の縁を描画する部分は次の 4 つに分けれられる. ただし角度は回転を行う前, つまり初期の角度である.

1. 外側の縁を描画する部分 (0 度から 130 度) : リスト 10 の 6〜19 行目
2. 外側の縁を描画する部分 (160 度から 270 度) : リスト 10 の 23〜36 行目
3. 内側の縁を描画する部分 (70 度から 200 度) : リスト 10 の 42〜55 行目
4. 内側の縁を描画する部分 (240 度から 380 度) : リスト 10 の 59〜72 行目

描画する位置や角度は値の変更によって容易に行えるから, ここでは「外側の縁を描画する部分 (0 度から 130 度)」について説明する. プログラムではリスト 10 の 6〜19 行目である. 時計の縁の描画は 2 つの扇型を描画することで行っている. まず,0 度から 130 度, 長さが 128 の扇型を描画する. 色はライトモードではライムグリーン, ダークモードではパープルである. 次に-1 度から 131 度, 長さが 124 の扇型を描画する. 色は背景色と同じ色を用いる. 最初に描いた扇型に, 背景色と同じで, 長さが短い扇型を上書きすることで, 図 1 で円の弧のような図形が描かれる. 上書きする扇形の角度を, 元の扇型の角度より大きくしているのは, 角度が同じであると端の線が表示される場合があるためである.

プログラムでは, まず,6 行目で長さを 128 に設定し,7 行目から 11 行目で色の設定を行っている. そして, 12 行目で drawDesign 関数を用いて色付きの扇型を描画している.drawDesign 関数の定義はリスト 10 の 89 行目から,99 行目である.drawDesign 関数は引数として, 中心座標 (xc,yc), 扇形を描く角度 thetaStart,thetaEnd, 長さ l, 扇形を回転させる変数 loop をとる. 扇形を描く角度は, 例として 0 度から 90 度の扇形を描く場合は,thetaStart に 0,thetaEnd に 90 を与える. drawDesign の内部では,GL_POLYGON および for 文を用いて扇形の描画を行っている.GL_POLYGON は指定された点を結ぶ図形を描く設定である.for 文では thetaStart から thetaEnd まで 1 度ずつ calPosition 関数で座標を計算して,glVertex2i 関数で描画を行っている.calPosition 関数の角度に theta+loop を与えることで図形全体が回転する仕組みになっている.

背景色と同じ扇形の描画も drawDesign 関数を用いることで同様に行う. これによって, 角度および長さを変更するだけで時計の縁を描画することができる. 他の 3 つの時計の縁も, 同様にして描画できる.

時計の縁全体を回転させる処理はリスト 10 の 75 行目から 83 行目で行っている. 変数 loop1,loop2 はリスト 1 でグローバル変数として定義されていた.Display 関数が呼ばれるたびに loop1 は $+0.05$,loop2 は -0.1 される. この変数を drawDesign 関数に与えることで図形全体が回転する.77 行目から 79 行目では,loop1 が 2π より大きい, つまり図形が一周したときに変数 loop1 を 0 に戻す処理を行っている. これを行うことで変数 loop1 が無限に大きくなることを防いでいる. 同様に,80 行目から 82 行目では変数 loop2 が -2π より小さくなったときに, 変数 loop2 を 0 に戻す処理を行っている.

5 ビルド方法の説明

ビルド方法について説明する. まず, ダウンロードした「j17406.tar.gz」を解凍する. 解凍は次のコマンドで行う.

```
gzip -dv j17406.tar.gz
tar xvf j17406.tar
```

解凍ができたからビルドを行う. 解凍したファイルは次のような構造になっている.

```
j17406
├── Makefile
├── j17406.c
├── myicon.o
├── myicon.rc
├── readme.txt
└── smalltile.ico
```

ビルドは次のコマンドで行う.「make」のみでコンパイルが終わる仕組みを説明する. ファイルの中には make でビルドを行うための Makefile が用意されている. Makefile のコードをリスト 11 に示す.

```
make
```

リスト 11 の 6 行目は最終的に出来上がるファイルを指定している. ここでは j17406.exe というファイルが出来上がる.9 行目ではコンパイルするファイルを指定している. コンパイルするファイルは j17406.c であるからこれを指定する.18 行目から 22 行目では, コンパイラ, リンカの指定, およびコンパイルオプションの指定を行っている. これらの設定を Makefile に書いておくことで,「make」と入力するだけでコンパイルが行われる.

36,37 行目では make コマンドにオプションをつけ「make clean」としたときの処理を記述している.「make clean」を実行すると,make を実行したときに生成される「.o」といったファイルが削除される.

リスト 11: Makefile

```
1  #
2  # Makefile
3  #
4
5  # 最終目的ファイル
6  TARGET = j17406.exe
7
8  # ソースファイル一覧(*.c)の一覧
9  SRCS = j17406.c
10
11 # オブジェクトファイル一覧(*.o)
12 OBJS = ${SRCS:.c=.o}
13
14 # ヘッダファイルの一覧(何もなし)
15 HEADERS =
16
17 # コンパイラ・リンカの指定
18 CC = gcc
19 CCFLAGS = -Wall -I/usr/include/opengl
20 LD = gcc
21 LDFLAGS =
22 LIBS = -lglpng -lglut32 -lglu32 -lopengl32 -lm myicon.o
23
24 #OBJSからTARGETを作る方法
25 $(TARGET) : $(OBJS)
26     $(LD) $(OBJS) $(LDFLAGS) -o $(TARGET) $(LIBS)
27
28 # *.cから*.oを作る方法
29 .c.o:
30     $(CC) $(CCFLAGS) -c $<
31
32 # *.oはHEADERSとMakefileに依存
33 $(OBJS) : $(HEADERS) Makefile
34
35 # make cleanとしたときに実行されるコマンド
36 clean :
37     rm -f $(TARGET) $(OBJS) core *~
```

6　実行結果とその説明

本章では, 次に示す実行結果およびその説明について述べる.
!付録