情報工学実験１
C言語応用

担当教員名:赤嶺有平
氏名:澤岨千明
学籍番号:0757230G

実験日:2008/06/06
提出日 2008/08/17

level 1.1

演習１のコードの実行フローを調べよ。（関数呼び出しを調べる。）

演習１のコード

```
#include <stdio.h>

void DoCallback( int (*cbfunc)(int,int,int) ) //コールバック関数を呼び出す関数
//cbfunc は int を返し，三つの int 型引数をとる関数へのポインタ
{
    int ret = cbfunc(0, 1, 2); //コールバック関数を呼び出す
    printf("callback function returned %d\n", ret);
}

int MyCallbackFunc1(int l, int c, int r)
{
    printf("MyCallbackFunc1 is called\n");
    return l+c+r;
}

int MyCallbackFunc2(int l, int c, int r)
{
    printf("MyCallbackFunc2 is called\n");
    return l-c-r;
}

int main()
{
    DoCallback(MyCallbackFunc1);
    DoCallback(MyCallbackFunc2);
}
```

考察

int main() の中の MyCallbackFunc1 によって、DoCallbak 関数を呼び出す。

MyCallbackFunc1 は DoCallbak の cbfunc() から引数を得て、それぞれ l=0，c=1，r=2 を渡されて実行制御を移す。

DoCallback 関数に実行結果 l+c+r = 3 を返す。この結果を ret に保存し、結果を printf で出力する。

MyCallbakFnc2 も同様な処理である。

level 1.2

演習 1.3 のプログラムに，最初の文字が"#"で始まる行を抽出するコールバック関数を追加し，そのコールバック関数を呼び出す様に main 関数を修正せよ.

callback3 #.c のソース

```
#include <stdio.h>
#include <string.h>
```

```
#define BUF_SIZE 256

/* fpで表現されるファイルからコールバック関数による条件判断により行を抽出する */
void searchWith(FILE* fp, int (*callback)(char* line) )
{
  char buf[BUF_SIZE];

  while(fgets(buf, BUF_SIZE, fp)) {
    if(callback(buf)) {
      printf("%s", buf);
    }
  }
}

/* 条件判断に使われるコールバック関数 */
int isContain(char* line) { /* line[0]=='#' */
  return strstr(line, "printf") != NULL;
  /* 上記は，以下のコードと同じ結果になる.
      if(strstr(line, "printf") != NULL)
        return 1;
      else
        return 0;
  */
}

/* "#"の検索に仕様するコールバック関数 */
int searchSharp(char* line)
{
  char* s; /* 最初の一文字 */
  s = strstr(line, "#");
  if (s == line)
    return 1;
  else
    return 0;
  /*  return strstr(line, "#") !=NULL; */
}

int main(void)
{
  searchWith(stdin, searchSharp);

  return 0;
}
```

実行結果

```
% ./a.out
#abc
#abc

abc#
^C
```

考察

#を検索のコールバック関数を書き換えた。最初の一文字を取得して、それが#なら出力するようになっている。

level 1.3

実際の開発現場では、ライセンスの関係等でライブラリの内部を変更できないことがある。
上記の例では、callback4bSub.c をライブラリと考えると、コールバック変数を用いる事で
callback4bMain.c だけを変更することで様々な抽出条件を指定することができている。
コールバック変数を使わずに同様の事（callback4bMain.c 内で抽出条件を指定する）
を実現する方法を考察せよ。

考察

新たにコールバック関数に関するライブラリを作成し、それをヘッダファイルとして読み込めば可能である。だが、リストをたどるたびに同様の関数を呼び出す必要があり、ヘッダファイルが増えるので、汎用ライブラリには良いものとはいえない。

level 1.4

コールバック関数の有用性について論ぜよ。

考察

コールバック関数とは他のコードの引数として渡される関数である。
コールバック関数は、呼び出し先で何らかのイベントが発生した場合の処理を指定する目的で使われることが多い。というのも、イベントの発生を検知するための処理と、そのイベントが起こった時に実行すべき個々の処理を分離して記述することができるからである。

level 2.1

glut.c の動作について考察せよ。特に、ユーザの操作に対して GLUT がどのタイミングで
どのコールバック関数を呼び出しているのか調べよ。

glut.c のソース

```
#include <GLUT/glut.h>
#include <stdio.h>

//描画イベントハンドラ
void display(void)
{
  printf("display\n");
}

//マウスイベントハンドラ
void mouse(int button, int state, int x, int y)
{
  if(state == GLUT_UP) {
    printf("mouse up\n");
  }else{
```

```
    printf("mouse down\n");
  }
}

//キーボードイベントハンドラ
void keyboard(unsigned char key, int x, int y)
{
  printf("keyboard(%c)\n", key);
}

int main(int argc, char *argv[])
{
  glutInit(&argc, argv); //glut の初期化
  glutCreateWindow(argv[0]); //window の作成

  //イベントハンドラの登録
  glutDisplayFunc(display);  //display コールバック関数の登録
  glutMouseFunc(mouse);      //mouse コールバック関数の登録
  glutKeyboardFunc(keyboard);//keyboard コールバック関数の登録

  glutMainLoop(); //イベントループ
  printf("exit\n");
  return 0;
}
```

考察

glut.c はユーザがイベントを起こす度に、それに対応するコールバック関数を呼び出している。イベントハンドラとはイベントが発生したときに特定の処理を与えるための記述である。

「glutMainLoop();」は GLUT プログラム内で一度だけコールし、返ってこない。だが、必要に応じて登録されたコールバック関数をコールする。

level 2.2

なぜイベント駆動型プログラムが必要とされているのか考察せよ。

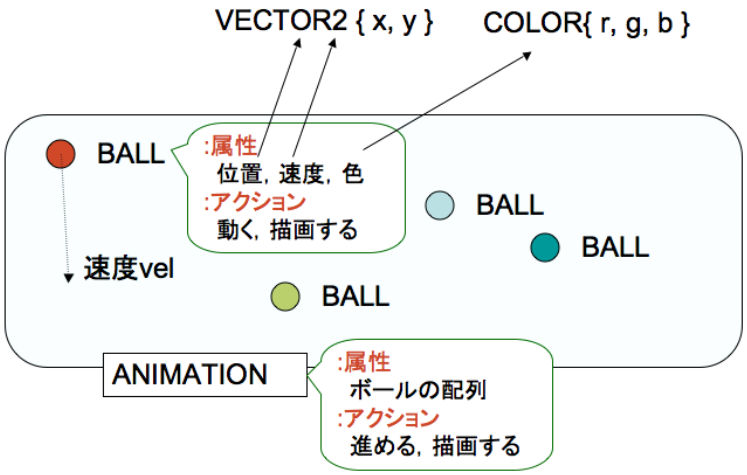
考察

ユーザが操作を行っていないときはプログラムは何もせず待機しているため、ユーザはそのプログラムを待たせた状態で他の操作を行うことができる。
プログラムを書く際に必要なイベントハンドラのみに処理を書けば良いということや、処理の記述をハンドラごとに分けるので、分かり易くなる。
過度にシステムに負荷を掛けるなどの望まないプログラムを減らす効果が期待できる。

level 3

上記、ラインアートのプログラムを参考に、ボールが画面を動き回るプログラムを作成せよ。
なお、ボールの動きは、以下の様に定義する。
・衝突しなければ、一定速度で移動し続ける
・他のボールと衝突すると跳ね返る
・ウィンドウの端にぶつかると跳ね返る

データ構造を下図で定義する。



balls.c のソース

```
/*
 *  balls.c
 *
 *
 *  Created by Yuhei Akamine on 08/06/03.
 *
 */
#include <OpenGL/gl.h>
#include <glut/glut.h>
#include <stdlib.h>

#include "balls.h"

//malloc ヘルパー関数．構造体用のメモリ領域を確保する
#define NEW(OBJ) ((OBJ*)malloc(sizeof(OBJ)))
#define NEW_ARRAY(OBJ, SIZ) ((OBJ*)malloc(sizeof(OBJ)*(SIZ)))

//定数定義
#define BALL_RAD 0.1 //ボールの半径

//0 以上 f 未満の実数乱数を返すヘルパー関数
float frand(float f)
{
    return (float)rand() / RAND_MAX * f;
}
```

```
//構造体 BALL を作成する
BALL makeBall(VECTOR2 pos, VECTOR2 vel, COLOR col)
{
  /* 追記した部分 */

  BALL p;
  p.pos = pos;
  p.vel = vel;
  p.col = col;

  return p;
}

//BALL を移動する
void moveBall(BALL* ball)
{
        //位置に速度を加算する
        /* 追記した部分 */
  ball->pos.x += ball->vel.x;
  ball->pos.y += ball->vel.y;
        //ウインドウの端で跳ね返る
        /* 追記した部分 */
  if(ball->pos.x < -1 || ball->pos.x > 1)
    ball->vel.x = -ball->vel.x;
  if(ball->pos.y < -1 || ball->pos.y > 1)
    ball->vel.y = -ball->vel.y;
}

//BALL を描画する
void drawBall(BALL* ball)
{
        glColor3f(ball->col.r, ball->col.g, ball->col.b); //色を指定する

        glPushMatrix();
        glTranslatef(ball->pos.x, ball->pos.y, 0);
        glutSolidSphere(BALL_RAD, 16, 2); //球体を描画する
        glPopMatrix();
}

//ANIMATION を生成する（領域を確保する）
ANIMATION* newAnimation(int num_balls) {

    ANIMATION* newAnim = NEW(ANIMATION);

    newAnim->num_balls = num_balls;
    newAnim->balls = NEW_ARRAY(BALL, num_balls);

    return newAnim;
}

//ANIMATION を進める
void forwardAnimation(ANIMATION* anim)
```

```
{
    int i,j;
    float dx,dy;  /* 新たな変数 */

    for(i=0; i<anim->num_balls; ++i) {
        BALL* b = anim->balls+i;
        moveBall(b);

                //衝突判定
                for(j=0; j<anim->num_balls; ++j) {
                        if(i!=j) {
                                BALL* b2 = anim->balls+j;

                                        /* 追加した部分 */
                          /* 物体 a と物体 b の距離が半径 (BALL_RAD) の 2 倍以下であれば
                          衝突している.
                          跳ね返りは，物体 a，b の速度を入れ替えることで実装できる */
                                dx = b2->pos.x - b->pos.x;
                                dy = b2->pos.y - b->pos.y;

                                /*BALL b と b2 の距離が BALL_RAD の 2 倍以下か？ */
                                if(dx*dx + dy*dy < (2*BALL_RAD)*(2*BALL_RAD)) {
                                    BALL *bb;

                                        //衝突した時の処理

                                        //色を変える
                                        b->col = makeColor(frand(1), frand(1), frand(1));

                                        //跳ね返り
                                        /* 追加した部分 */
                                        bb->vel = b->vel;
                                        b->vel = b2->vel;
                                        b2->vel = bb->vel;

                                }
                        }
                }
        }
}
//ANIMATION を描画する
void drawAnimation(ANIMATION* anim)
{
    int i;

    for(i=0; i<anim->num_balls; ++i) {
        BALL* b = anim->balls+i;
        drawBall(b);
        }
}
```

考察

「makeBall」では構造体 BALL の作成。
p.pos はボールの位置、 p.vel にはボールの速度、p.col にはボールの色である。

「moveBall」では BALL の移動。
ball->pos.x += ball->vel.x;、ball->pos.y += ball->vel.y; で今のボール位置にそれぞれ x 方向、y 方向の速度を加算している。

```
if(ball->pos.x < -1 || ball->pos.x > 1)    ball->vel.x = -ball->vel.x;
if(ball->pos.y < -1 || ball->pos.y > 1)    ball->vel.y = -ball->vel.y;
```

で、ボールが画面端にくると、逆の速度を代入して跳ね返るようにしている。

「forwardAnimation」では ANIMATION を進めている。
この中で衝突時の処理を書いているのだが、これに新たな変数 float int dx, dy を追加している。dx に b と b2 の x 軸方向の距離、 dy には y 軸方向の距離を代入し、始めのほうで定義してあるボールの半径 BALL_LAD の二乗より小さいかどうかで衝突しているか否かを判定している。
実際の跳ね返る記述はもう一つ準備していた bb という変数に b の速度を代入した後、b に b2 の速度を入れ最後に b2 に bb の速度を代入することで二つの速度を入れ替えている。

参考

- コールバック関数【callback function】
  http://ew.hitachi-system.co.jp/w/E382B3E383BCE383ABE38390E38383E382AFE996A2E695B0.html
- イベント駆動型プログラミング - Wikipedia
  http://ja.wikipedia.org/wiki/イベント駆動型プログラミング