C言語利用環境の開発

The Development of the exploiting environments for C language

小　林　信　雄

Nobuo Kobayashi

1．まえがき

コンピュータ言語は種々のものが使用されているが、中でも近年注目されているがC言語であろう。特に1989年 ANSI 規格として制定されて以来、急速に言語仕様の標準化が加速された。マイクロ、ミニ、EWS クラスのコンピュータにおいては主要言語として使用されており、メイン・フレームやスーパーコンピュータにも移植されつつある。更に最近大きく取り上げられているオブジェクト指向のプログラミングの潮流に対してもC＋＋や Objective C などが開発され、一層C言語への関心が高まりつつある。

このようなコンピュータ言語の変化に対応すべく、本学においてもコンピュータ教育にC言語を取り入れる試みがなされている。本学の教育用コンピュータとしては NEC 製PC9801シリーズと日立製2020とがある。PC9801シリーズには優れたコンパイラが容易に入手できるので環境的には整っている。しかしながら、2020にはサードパーティからの専用ソフトウェアの供給も少なく、現在は大型コンピュータの端末としてのみ使用されている。2020もハードウェア的には決して劣っているわけではないので、環境を整備すれば充分利用できるはずである。

このような観点から2020の使用環境を調査したところ、Cコンパイラとしては PC9801用に導入したものが使用できることが判明した。またプログラム作成には必須のエディタについても2020用のものが利用できるようになった。最後に残されたのはグラフィック環境の実現である。グラフィック環境は極度にハードウェアに依存するので、他機種のものを流用することは不可能である。2020には本来優れたグラフィック・システムが搭載されているので、これをC言語から利用できれば、2020の利用価値が大きくなる。

本研究においては、2020用のグラフィック機能をC言語関数として利用できるシステムの構築を目的とする。

2．HITACHI 2020のグラフィック環境

2020の CPU は80286であり、メモリは1 MB 実装されている。マルチタスクオペレーティングシステムのもとで TSS オンライン制御、MS-DOS、プリンタスプーラのマルチジョブを行っている。MS-DOSのバージョンは3.10であり、使用可能メモリは512KB である。

平成3年5月28日原稿受理

大阪産業大学　工学部

2020に使用されているグラフィックコントローラはACRTC HD63484である。画面構成は、画面分解能が1120＊720ドット、RGB3フレームからなり、色分解能はRGB各1ビットの8色である。ビデオRAMは図形画面がある。スクリーンにはこの4つのフレームが合成されて表示される。文字画面が図形画面に優先する。

アプリケーションからはハードウェアを直接操作することは許されず、ソフトウェア割り込みにるシステムコールでアクセスする。これは2020がホストコンピュータの端末としての動作とMS-DOSマシンとしての動作のマルチジョブを行っているため両者からの要求を調停する必要があるためである。

3．グラフィック用システムコール機能

グラフィック表示制御はソフトウェア割り込み51Hにより起動される。

3.1　コーリング・シーケンス

コーリング・シーケンスは次のとうりである。

AH＝ファンクションコール番号

DS：BX＝機能テーブルアドレス

INT 51H

全てのレジスタは保存される。

3.2　ファンクションコール機能

使用するファンクションコールは次のものである。

AX＝09H, DS：BX＝機能テーブルアドレス　フレームの確保/解放

AX＝09H, DS：BX＝機能テーブルアドレス　直接ドライバ起動ルーチン

3.2.1　フレームの確保

グラフィク機能の使用に先だってフレームバッファ用に領域を確保しておかねばならない。機能テーブルの内容を表3.1に示す。入力パラメータはオフセット0から8までであり、その他はドライバが設定する。画面種類としてはここではドットプレーンを確保するため4を指定する。その他の入力パラメータは0とする。ドライバから帰されるスクリーン番号はグラフィックルーチンとのインターフェイスに使用される。リターン値は0が正常終了でありそれ以外は領域は確保されていない。

3.2.2　フレームの解放

グラフィック機能の終了に際しては確保したフレーム用領域を解放しなければならない。機能テーブルの内容を表3.2に示す。入力パラメータはオフセット0から10までであり、その他はドライバが使用する。画面種類としてはここではドットプレーンを解放するため4を指定する。スクリーン番号はフレームの確保に際してドライバから帰された番号を指定する。その他の入力パラメータは0とする。リターン値は0が正常終了でありそれ以外は領域は解放されていない。

3.2.3　直接ドライバ起動

あらかじめ作成したDevice Control Block(DCB)を入力情報として直接に入出力制御ルーチンを起動する。機能テーブルの内容を表3.3に示す。入力パラメータはオフセット0から

表３．１　Format of Function Table to get console control (GCON)

オフセット	意味	
００	３３Ｈ	０６Ｈ
０２	画面種類	コード体系
０４	カーソル表示形態	
０６	ポップアップメニュサイズ	
０８	表示先左上アドレス	
１０	スクリーン番号	画面種類
１２	画面バッファオフセット	
１４	画面バッファセグメント	
１６	ATRバッファオフセット	
１８	ATRバッファセグメント	
２０	画面テーブルオフセット	
２２	画面テーブルセグメント	
２４		
２６		
２８		
３０		
３２	エラーステータス	
３４	リターン値	

表３．２　Format of Function Table to releace console control (RCON)

オフセット	意味	
００	３３Ｈ	０７Ｈ
０２	画面種類	
１０	スクリーン番号	
３４	リターン値	

表３．３　Format of Function Table to call the driver directly (CALLMAC)

オフセット	意味	
００	２５Ｈ	００Ｈ
０２	ＤＣＢオフセット	
０４	ＤＣＢセグメント	
０６	バッファセグメント	
０８	００００Ｈ	
１０	００００Ｈ	
１２	リターン値	

10までであり、その他はドライバが使用する。リターン値は０が正常終了でありそれ以外は異常終了である。グラフィックドライバの起動以前にフレーム確保を実行しておかねばならない。

３．３　DCB ファンクション

本グラフィックドライバが認識する DCB ファンクションは次の８種類である。

１．GSC write　　GSC(Graphic Sub Command)列を処理する。(描画系)
２．Paint 1　　閉領域の塗りつぶし。
３．Read Graphic Plane　　フレームバッファのドットデータの読み込み
４．Write Graphic Plane　　フレームバッファのドットデータの書き込み
５．Graphic Display Control　　グラフィック画面の表示／非表示
６．Paint 2　　閉領域の塗りつぶし。
７．Set Pallet　　カラーパレットの色指定
８．Reset Pallet　　カラーパレットの色指定をデフォルトに戻す。

DCB のフォーマットを表3.4に示す。

各 DCB ファンクションの呼び出しに必要な DCB の項目はそれぞれ異なっている。C 言語による DCB の定義及び各 DCB ファンクションの呼び出しを図3.5, 3.6に示す。

表3.4 Format of Device Control Block (DCB)

オフセット	名称	内容
0	起動先ドライバ番号	
1	起動元ドライバ番号	
2	終了コード	
3	終了ステータス	
4	未使用	
5	ファンクション	
6－7	未使用	
8－9	装置情報	8=スクリーン番号
10	ステーションアドレス	競合時
11	ジョブ番号	競合時
12	ステーションアドレス	起動元
13	ジョブ番号	起動元
14－15	未使用	
16－19	バッファアドレス	オフセット、セグメント
20－21	バッファ長さ	
22－23	バッファ実行長さ	
24－27	サブバッファアドレス	オフセット、セグメント
28－29	サブバッファ長さ	
30－31	サブバッファ実行長さ	
32	IDコード	
33	Multi Use	
34	カラー	
35－38	物理座標	X、Y
39－40	GSCカウンタ	
41－44	物理サイズ	X、Y
45－48	Rest	
49－63	未使用	

表3．5　Definition of Structure in C Syntax to represent DCB

```
/*************************/
/* device control block */
/*************************/
struct gdrv_dcb2 {      /*****************************************/
char    task_num;       /* ØØ CGDRV task no (Ø7h)               */
char    owner_tasknum;  /* Ø1 fixed 8Øh                         */
char    comp_code;      /* Ø2 completion code (8Øh/cØh)         */
char    err_stat;       /* Ø3 error status                      */
char    padØ;           /* Ø4                                   */
char    func;           /* Ø5                                   */
short   pad1;           /* Ø6                                   */
short   scr_num;        /* Ø8 screen number                     */
char    stationØ;       /* 1Ø station address                   */
char    jobnumØ;        /* 11 job number                        */
char    owner_station;  /* 12 owner station adrs                 */
char    owner_jobnum;   /* 13 owner job number (Ø1)              */
short   pad2;           /* 14                                   */
char far *buffer;       /* 16-19 buffer offset                  */
short   buf_len;        /* 2Ø buffer length                     */
short   exec_len;       /* 22 executed length                   */
char far *subbuffer;    /* 24-27 sub buf offset                 */
short   subbuf_len;     /* 28 sub buf length                    */
short   subexec_len;    /* 3Ø sub buf len execed                */
char    id_code;        /* 32 id code                           */
char    multi_use;      /* 33                                   */
char    color;          /* 34                                   */
short   xcood;          /* 35 x coordinate                      */
short   ycood;          /* 37 y coordinate                      */
short   gsc_cnt;        /* 39 gsc counter                       */
short   xsize;          /* 41 physical x size                   */
short   ysize;          /* 43 physical y size                   */
short   rest1;          /* 45                                   */
char    rest2;          /* 47                                   */
char    rest3;          /* 48                                   */
char    nouse[15];      /* 49-63                                */
};                      /*****************************************/
```

表3.6 Function Definition to call DCB function of GSC-write

```
/*[NAME]
 *      GSCwrite                write graphic sub command
 *[SYNTAX]
 *      int far GSCwrite1(cmdbuf, cmdlen)
 *      char    *cmdbuf
 *      int     cmdlen          byte length
 *
 *      return  Ø                       if ok
 *[DESCRIPTIONS]
 *[END]
 */
int far GSCwrite1(char far *cmdbuf, int cmdlen)
{
        int     rtnc;
        GDCB2   *gptr;

        gptr = &g_dcb2;
        clr_gdcb( gptr );
/* common data */
        gptr->task_num     = Øx Ø7;     /* CGDRV task number (fixed)    */
        gptr->owner_tasknum= Øx8Ø;      /* fixed                        */
        gptr->owner_jobnum = Øx Ø1;     /* fixed                        */
        gptr->scr_num      = screenNum;
/* gsc write */
        gptr->func       = Øx9Ø;        /* func code for GSC write      */
        gptr->buffer     = cmdbuf;      /* cmd buffer address           */
        gptr->buf_len    = cmdlen;      /* cmd buffer length            */
        gptr->multi_use  = Ø;           /* fixed                        */
        gptr->gsc_cnt    = cmdlen;      /*                              */
        gptr->subbuf_len = Ø;           /* fixed                        */
        if( (rtnc=gdrvr3(gptr)) != Ø ){
                return(rtnc);
        }
        if( (gptr->comp_code & Øxc Ø) == Øxc Ø) {
                return(-1);
        }
        return(Ø);
}
```

4．グラフィックC関数

DCBファンクションにより実現されたグラフィックC関数の機能を以下に示す。

4.1　初期化、終了処理

4.1.1　int gopen(void)

グラフィック機能の使用に先だって一度だけ実行されなければならない。フレームバッファの確保、グラフィックパラメータの初期化、グラフィック画面のクリアを行う。戻り値が0以外の場合には、以後の動作は保証されないのでプログラムの実行を終了させねばならない。

4.1.2　int gclose(void)

グラフィック機能の使用終了に伴う後処理を行う。プログラムの最後に一度だけ呼び出す。戻り値が0以外の場合には異常終了である。

4.2　描画関数

描画関数は、ワールド座標系ではwで始まり、ビュウー座標系ではvで始まる。関数の戻り値は正常終了の場合には0である。

4.2.1　int wDrawPoly (int n, double*x, double*y, int color)

ワールド座標系でn点の(x, y)座標を結ぶ線分を色コードcolorで描く。

参照：setPF, setLineType

4.2.2　int wDot (double x, double y, int color)

ワールド座標系での点(x, y)に色コードcolorの点を描く。

参照：setPF

4.2.3　int wLine (double x0, double y0, double x1, double y1, int color)
　　int vLine (in x0, int y0, int x1, int y1, int color)

2点(x0, y0)，(x1, y1)を結ぶ線分を色コードcolorで描く。

参照：setPF, setLineType

4.2.4　int wRect (double x0, double y0, double x1, double y1, int color)
　　int vRect (int x0, int y0, int x1, int y1, int color)

2点(x0, y0), (x1, y1)を対角線とする長方形を色コードcolorで描く。

参照：setPF, setLineType

4.2.5　int wCircle (double x, double y, double r, int color)
　　int wCircle (int x, int y, int r, int color)

(x, y)を中心として、半径rの円を描く。

参照：setPF, setLineType

4.2.6　int wArc (double x, double y, double r, double sangle, int color, int fcntl)

(x, y)を中心、半径rの円弧を角度sangleからeangle迄、色コードcolorで描く。

fcntl　＝0　円弧
　　＝1　円弧に中心から開始点まで直線で結ぶ。
　　＝2　円弧に中心から終了点で結ぶ。
　　＝3　扇形

参照：setPF, setLineType

4.2.7　int　wEllipticarc (double x, double y, double rx, double ry, double sangle, double

eangle, int color, int fcntl)

(x, y)を中心、x 半径 rx, y 楕円弧を角度 sangle から eangle 迄、色コード color で描く。fcntl は wArc を参照。参照：setPF, setLineType

4．2．8　int wEllipse (double x, double y, double rx, double ry, int color, int fcntl)

(xb y)を中心、x 半径 rxb y 半径 ry の楕円を色コード color で描く。fcntl は wArc を参照。

参照：setPF, setLineType

4．3　描画属性の制御

図形表示の際に表示属性を変更することが出来る。

4．3．1　カラー属性

int setVectorColor (int color)　以降の表示色を指定する。

int getVectorColor (void)　現在の表示色を得る。

参照：表4.1

4．3．2　線種属性

int setLineType (int type)　以降の表示線種を指定する。

int getLineType (void)　現在の表示線種を得る。

参照：表4.2

表4．1　Color Code

Color	Code
black	0
blue	1
green	2
cian	3
red	4
magenta	5
yellow	6
white	7

表4．2　Line Type

線　種	記　号	コード
実　線	LTP_fill	0
短点線	LTP_shortdot	1
長点線	LTP_longdot	2
長破線	LTP_longbroken	3
短破線	LTP_shortbroken	4
1点鎖線	LTP_chain1	5
2点鎖線	LTP_chain2	6
セルパターン	LTP_cell	7

4．3．3　ペイント方式

int setPF (int pf)　以降のペイント方式を指定する。

int getPF (void)　現在のペイント方式を得る。

参照：表4.3

表4．3　Paint Format

ペイント方式	描　画　方　法	記号	コード
オーバー	パターン中1は指定色で、0は元の画面	PF_over	0
ミックス	パターン中1は元の色と指定色の混合色で、0は元の画面	PF_mix	1
アンダー	パターン中1で元が0の部分のみ指定色で表示	PF_under	2
コンプリメント	パターン中1の部分に対応する元の画面の0、1を入れ換える	PF_comp	3
リプレース	パターン中1は指定色で、0は黒色で表示	PF_repl	4

4．3．4　ユーザー定義パターン

int setUserPattern (int pattern)　ユーザー定義パターンを指定する。

int getUserPattern (void)　　現在のユーザー定義パターンを得る。

4．3．5　VFパラメータ

各コマンドにより意味が異なる。

int setVF (int val)　VFパラメータを設定する。

int getVF (void)　現在のVFパラメータを得る。

4．4　ピクセルアクセス

フレームバッファのピクセル単位の読み書きを行う。

4．4．1　int wPutPixel (double x, double y, int color, int func)

int wGetPixel (double x, double y)

int vPutPixel (int x, int y, int color, int func)

int vGetPixel (int x, int y)

(x, y)のピクセルデータを得る。(GetPixel)

(x, y)のピクセルを色コード color で書き換える。(PutPixel)

func　＝0　　置換する。

＝1　　元のデータの論理否定をとる。(color は使用されない)

＝2　　元のデータとの論理積をとる。

＝3　　元のデータとの論理和をとる。

＝4　　元のデータとの排他的論理和をとる。

4．5　ペイント

閉領域を形成する線分が複雑すぎる場合には処理を中止する。本ドライバでは黒色を特別扱いしているため境界色が黒色かそれ以外かで区別する必要がある。ペイント方式については表4.3を参照。

4．5．1　int wPaint (double x, double y, int color, int edgecolor)

(x, y)を含む色 edgecolor で囲まれた領域を色コード color で塗りつぶす。ペイント方式はオーバーである。境界色 edgecolor は0 (Black)以外とする。

4．5．2　int wPaintBack (double x, double y, int color)

(x, y)を含む黒色以外で囲まれた領域を色コード color で塗りつぶす。種々のペイント方式、

ペイントパターンを選択できる。

参照：setShadeA, setImageFill

4．5．3　int wRepaint (double x, double y)

(x, y)を含む黒色で囲まれた領域を色コードcolorで塗りつぶす。ペイントの色、ペイント方式、ペイントパターンをあらかじめ設定しておく。

参照：setShadeA, setImageFill

4．5．4　int wPaint2 (double x, double y, int color, int edecolor)

(x, y)を含む色edgecolorで囲まれた領域を色コードcolorで塗りつぶす。ペイント方式はリプレースのみである。この場合には境界色に制限はない。

4．6　ペイント表示属性の制御

以下の機能はペイント関数のうちwPaintBack, wRepaintで使用できる。

4．6．1　setShadeA (int pf, int color, int pptn)

pf　　ペイント方式（表4.3）

color　ペイントカラー（表4.1）

pptn　ペイントパターンコード（0x20, 0x1A，others）

4．6．2　setImageFill (int color)

ペイントパターンタイルとして均一な色タイルを使用する。setShadeAのペイントパターンコード0x1Aとして参照することが出来る。

4．6．3　setImagel (int pf, int color, char＊datap)

pf　　ペイント方式（表4.3）

color　ペイントカラー（表4.1）

datap　8バイト長のデータ領域へのポインタ

datapでポイントされる8バイトを8＊8ビットパターンとみなして、これをペイントパターンタイルとして使用する。setShadeAのペイントパターンコード0x1Aとして参照することが出来る。

4．7　シェイディング

4．7．1　int wFilledBox (double x0, double y0, double x1, double y1, int color)

int vFilledBox (int x0, int y0, int x1, int y1, int color)

2点(x0, y0), (x1, y1)を対角線とする長方形を色コードcolorで埋める。

パラメータ　VF＝0　枠線なし

＝1　枠線あり

参照：setVF, setPF, setLineType

4．7．2　int wBoxClear (double x0, double y0, double x1, double y1)

int vBoxClear (int x0, int y0, int x1, int y1)

2点(x0, y0), (x1, y1)を対角線とする長方形内部を消去する。

4．8　文字表示

グラフィックプレーンに文字を表示する。表示される文字の大きさは24＊24ドットである。

4．8．1　int vcharacter (int x, int y, int leng, char＊str)

座標(x, y)を左下端として水平方向に文字を表示する。

4．8．2　int vcharacter (int x, int y, int ieng, char＊str)

座標(x, y)を最初の文字の左下端として垂直方向に文字を表示する。

4．9　文字表示属性の制御

4．9．1　int setChrPF (int pf)　　水平表示文字のペイント形式を設定する。

int getChrPF (void)　　現在の水平表示文字のペイント形式を得る。

4．9．2　int setChrColor (int color)　　水平表示文字の色を設定する。

int getChrColor (void)　　現在の水平表示文字の色を得る。

4．9．3　int setChrScale (int scale)　　水平表示文字の大きさを設定する。

int getChrScale (void)　　現在の水平表示文字の大きさを得る。

scale＝0，1，2　　1倍，4倍，1/4倍

1/4倍は半角コードのみ。

4．9．4　int setChrRotate (int angle)　　水平表示文字の回転を設定する。

int getChrRotate (void)　　現在の水平表示文字の回転角を得る。

angle＝0，1，2　　0度，90度，-90度

4．9．5　int setVChrPF (int pf)　　垂直表示文字のペイント形式を設定する。

int getVChrPF (void)　　現在の垂直表示文字のペイント形成を得る。

4．9．6　int setVChrColor (int color)　　垂直表示文字の色を設定する。

int getVChrColor (void)　　現在の垂直表示文字の色を得る。

4．9．7　int setVChrScale (int scale)　　垂直表示文字の大きさを設定する。

int getVChrScale (void)　　現在の垂直表示文字の大きさを得る。

scale＝0，1，2　　1倍，4倍，1/4倍

1/4倍は半角コードのみ。

4．9．8　int setVChrRotate (int angle)　　垂直表示文字の回転角を設定する。

int getVChrRotate (void)　　現在の垂直表示文字の回転角を得る。

angle＝0，1，2　　0度，90度，-90度

4．10　座標系

物理座標、論理座標、ワールド座標間の変換を行う。

4．10．1　int setViewPort (int xll, int yll, double xrh, double　yrh)

物理座標の左下(xll, yll)と右上(xrh, yrh)を描画範囲とする。論理座標は物理座標とおなじである。

4．10．2　int setWorld (double xll, double yll, double xrh, double yrh)

setViewPort で決めた描画範囲をワールド座標の左下(xll, yll)と右上(xrh, yrh)に射影する。

4．10．3　double getAspect (void)

描画範囲のアスペクト比を求める。

4．11　その他

4．11．1　int gfrmclr (void)

グラフィック画面の消去。グラフィックパラメータも初期化される。

5　結論

開発したグラフィックC関数は現在、工学部において実験・実習、演習、卒業研究に使用されている。グラフィカル表現は、結果の見通しが良くなるだけでなく、学生の興味を引き付けるにも一定の効果を挙げている。

今後の課題としては

（1）処理速度の向上

一部の低レベルルーチンはアセンブラで書き換えて、最適化を図っているが、一層の改善のためにはコプロセッサの導入も考慮する必要がある。

（2）他言語とのリンク

グラフィック機能の必要性は他の言語、例えばFortran, Pascal等についても同様であろうから、これらとのリンクが出来ることが望ましい。これは、パラメータの受渡し方法の違いを調整するだけで対応できると思われる。

（3）グラフィック端末化

メイン・コンピュータとの交信が可能であるので、本機をグラフィック端末として、使用することが可能である。

今後これらの点を勘案しながら機能の充実を図って行く予定である。

なお、本研究は平成2年度大阪産業大学産業研究所の特別研究費の助成を受けて行われたものです。

≪参考文献≫

1）日立パーソナルワークステーション2020解説書.

2）HD63484ユーザーズマニュアル、日立製作所、1987.