Podstawowa obsługa plików

Istnieją dwie metody obsługi czytania i pisania do plików:

- wysokopoziomowa,
- niskopoziomowa.

Nazwy funkcji z pierwszej grupy zaczynają się od litery "f" (np. fopen(), fread(), fclose()), a identyfikatorem pliku jest wskaźnik na strukturę typu FILE. Owa struktura to pewna grupa zmiennych, która przechowuje dane o pliku - jak na przykład aktualną pozycję w nim. Szczegółami nie musisz się przejmować, funkcje biblioteki standardowej same zajmują się wykorzystaniem struktury FILE. Programista może więc zapomnieć, czym tak naprawdę jest struktura FILE i traktować taką zmienną jako "uchwyt", identyfikator pliku.

Druga grupa to funkcje typu read(), open(), write() i close().

Podstawowym identyfikatorem pliku jest liczba całkowita, która jednoznacznie identyfikuje dany plik w systemie operacyjnym. Liczba ta w systemach typu UNIX jest nazywana **deskryptorem** pliku.

Należy pamiętać, że nie wolno nam używać funkcji z obu tych grup jednocześnie w stosunku do jednego, otwartego pliku, tzn. nie można najpierw otworzyć pliku za pomocą fopen(), a następnie odczytywać danych z tego samego pliku za pomocą read().

Czym różnią się oba podejścia do obsługi plików? Otóż metoda wysokopoziomowa ma swój własny bufor, w którym znajdują się dane po odczytaniu z dysku a przed wysłaniem ich do programu użytkownika. W przypadku funkcji niskopoziomowych dane kopiowane są bezpośrednio z pliku do pamięci programu. W praktyce używanie funkcji wysokopoziomowych jest prostsze a przy czytaniu danych małymi porcjami również często szybsze i właśnie ten model zostanie tutaj zaprezentowany.

Dane znakowe

Skupimy się teraz na najprostszym z możliwych zagadnień - zapisie i odczycie pojedynczych znaków oraz całych łańcuchów.

Napiszmy zatem nasz pierwszy program, który stworzy plik "test.txt" i umieści w nim tekst "Hello world":

```
#include <stdlib.h>
#include <stdio.h>
int main ()
{
	FILE *fp; /* używamy metody wysokopoziomowej - musimy mieć zatem identyfikator pliku, uwaga na gwiazdkę! */
	char tekst[] = "Hello world";
	if ((fp=fopen("test.txt", "w"))==NULL) {
		printf ("Nie mogę otworzyć pliku test.txt do zapisu!\n");
		exit(1);
	}
	fprintf (fp, "%s", tekst); /* zapisz nasz łańcuch w pliku */
	fclose (fp); /* zamknij plik */
	return 0;
}
```

Teraz omówimy najważniejsze elementy programu. Jak już było wspomniane wyżej, do identyfikacji pliku używa się wskaźnika na strukturę `FILE` (czyli `FILE *`). Funkcja **fopen** zwraca ów wskaźnik w przypadku poprawnego otwarcia pliku, bądź też NULL, gdy plik nie może zostać otwarty. Pierwszy argument funkcji to nazwa pliku, natomiast drugi to '**tryb dostępu** - **w** oznacza "write" (pisanie). Zwrócony "uchwyt" do pliku będzie mógł być wykorzystany jedynie w funkcjach zapisujących dane. I odwrotnie, gdy otworzymy plik podając tryb **r** ("read", czytanie), będzie można z niego jedynie czytać dane. Jak zatem uprościć nazwę typu `FILE*`? Używając typedef:

```
typedef FILE* plik;
plik fp;
```

Po zakończeniu korzystania z pliku należy plik zamknąć. Robi się to za pomocą funkcji fclose. Jeśli zapomnimy o zamknięciu pliku, wszystkie dokonane w nim zmiany zostaną utracone!

Pliki a strumienie

Można zauważyć, że do zapisu do pliku używamy funkcji `fprintf`, która wygląda bardzo podobnie do `printf` - jedyną różnicą jest to, że w `fprintf` musimy jako pierwszy argument podać identyfikator pliku. Nie jest to przypadek - obie funkcje tak naprawdę robią to samo. Używana do wczytywania danych z klawiatury funkcja `scanf` też ma swój odpowiednik wśród funkcji operujących na plikach - jak nietrudno zgadnąć, nosi ona nazwę `fscanf`.

W rzeczywistości język C traktuje tak samo klawiaturę i plik - są to źródła danych, podobnie jak ekran i plik, do których można dane kierować. Jest to myślenie typowe dla systemów typu UNIX, jednak dla użytkowników przyzwyczajonych do systemu Windows albo języków typu Pascal może być to co najmniej dziwne. Nie da się ukryć, że między klawiaturą i plikiem na dysku zachodzą podstawowe różnice i dostęp do nich odbywa się inaczej - jednak funkcje języka C pozwalają nam o tym zapomnieć i same zajmują się szczegółami technicznymi. Z punktu widzenia programisty, urządzenia te sprowadzają się do nadanego im identyfikatora. Uogólnione pliki nazywa się w C **strumieniami**.

Każdy program w momencie uruchomienia "otrzymuje" od razu trzy otwarte strumienie:

- **stdin** (wejście) = odczytywanie danych wpisywanych przez użytkownika
- **stdout** (wyjście) = wyprowadzania informacji dla użytkownika
- **stderr** (wyjście błędów) = powiadamiania o błędach

Warunki korzystania ze standardowych strumieni :

- dołączyć plik nagłówkowy stdio.h
- nie musimy otwierać ani zamykać strumieni standardowych (tak jak w przypadku niestandardowych plików : fopen i fclose)

Warto tutaj zauważyć, że konstrukcja:

```
fprintf (stdout, "Hej, ja działam!");
```

jest równoważna konstrukcji:

```
printf ("Hej, ja działam!");
```

Podobnie jest z funkcją scanf().

```
fscanf (stdin, "%d", &zmienna);
```

działa tak samo jak:

```
scanf("%d", &zmienna);
```

Obsługa błędów

Jeśli nastąpił błąd, możemy się dowiedzieć o jego przyczynie na podstawie zmiennej errno zadeklarowanej w pliku nagłówkowym errno.h. Możliwe jest też wydrukowanie komunikatu o błędzie za pomocą funkcji perror. Na przykład używając:

```
 fp = fopen ("tego pliku nie ma", "r");
 if( fp == NULL )
     {
     perror("błąd otwarcia pliku");
     exit(-10);
     }
```

dostaniemy komunikat:

```
błąd otwarcia pliku: No such file or directory
```

Inny sposób :

```
#include<stdio.h>
#include <errno.h>
int main()
{
errno = 0;
FILE *fb = fopen("/home/jeegar/filename","r");
if(fb==NULL)
        printf("its null");
else
        printf("working");
printf("Error %d \n", errno);
}
```

Zaawansowane operacje

Pora na kolejny, tym razem bardziej złożony przykład. Oto krótki program, który swoje wejście zapisuje do pliku o nazwie podanej w linii poleceń:

```
#include <stdio.h>
#include <stdlib.h>
/* program udający bardzo prymitywną wersję programu "tee" */
int main (int argc, char **argv)
{
     FILE *fp;
     int c;
     if (argc < 2)
     {
```

```
            fprintf (stderr, "Uzycie: %s nazwa_pliku\n", argv[0]);
            exit(-1);
    }
    fp = fopen (argv[1], "w");
    if (!fp)
    {
            fprintf (stderr, "Nie moge otworzyc pliku %s\n", argv[1]);
            exit(-1);
    }
    printf("Wcisnij Ctrl+D+Enter lub Ctrl+Z+Enter aby zakonczyc\n");
    while ((c = fgetc(stdin)) != EOF)
    {
            fputc(c, stdout);
            fputc(c, fp);
    }
    fclose(fp);
    return 0;
}
```

Tym razem skorzystaliśmy już z dużo większego repertuaru funkcji. Między innymi można zauważyć tutaj funkcję fputc(), która umieszcza pojedynczy znak w pliku. Ponadto w wyżej zaprezentowanym programie została użyta stała EOF, która reprezentuje koniec pliku (ang. *End Of File*). Powyższy program otwiera plik, którego nazwa przekazywana jest jako pierwszy argument programu, a następnie kopiuje dane z wejścia programu (*stdin*) na wyjście (*stdout*) oraz do utworzonego pliku (identyfikowanego za pomocą fp). Program robi to tak długo, aż naciśniemy kombinację klawiszy Ctrl+D (w systemach Unixowych) lub Ctrl+Z(w Windows), która wyśle do programu informację, że skończyliśmy wpisywać dane. Program wyjdzie wtedy z pętli i zamknie utworzony plik.

Rozmiar pliku

Dzięki standardowym funkcjom języka C możemy m.in. określić długość pliku. Do tego celu służą funkcje fsetpos, fgetpos oraz fseek. Ponieważ przy każdym odczycie/zapisie z/do pliku wskaźnik niejako "przesuwa" się o liczbę przeczytanych/zapisanych bajtów. Możemy jednak ustawić wskaźnik w dowolnie wybranym miejscu. Do tego właśnie służą wyżej wymienione funkcje. Aby odczytać rozmiar pliku powinniśmy ustawić nasz wskaźnik na koniec pliku, po czym odczytać ile bajtów od początku pliku się znajdujemy. Użyjemy do tego tylko dwóch funkcji: fseek oraz fgetpos. Pierwsza służy do ustawiania wskaźnika na odpowiedniej pozycji w pliku, a druga do odczytywania na którym bajcie pliku znajduje się wskaźnik. Kod, który określa rozmiar pliku znajduje się tutaj:

```
#include <stdio.h>
int main (int argc, char **argv)
{
   FILE *fp = NULL;
   fpos_t dlugosc;
   if (argc != 2) {
      printf ("Użycie: %s <nazwa pliku>\n", argv[0]);
      return 1;
      }
   if ((fp=fopen(argv[1], "rb"))==NULL) {
      printf ("Błąd otwarcia pliku: %s!\n", argv[1]);
      return 1;
      }
   fseek (fp, 0, SEEK_END); /* ustawiamy wskaźnik na koniec pliku */
   fgetpos (fp, &dlugosc);
   printf ("Rozmiar pliku: %d\n", dlugosc);
```

```
    fclose (fp);
    return 0;
}
```

Znajomość rozmiaru pliku przydaje się w wielu różnych sytuacjach, więc dobrze przeanalizuj przykład!

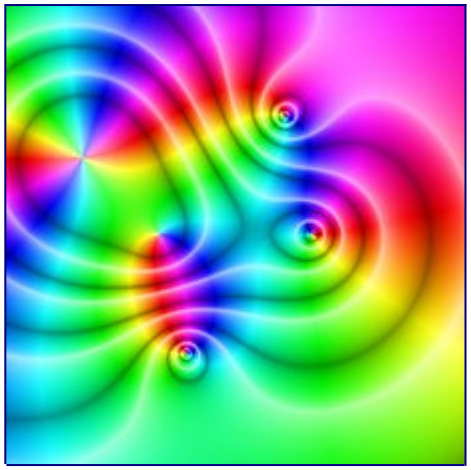
Przykład - pliki graficzne

Plik graficzny tworzymy :

- bezpośrednio w C (fprintf / fwrite)
- pośrednio za pomocą strumieni i potoku, wtedy :
    - zamiast komend zapisu do pliku (np. fprintf) używamy komend wysyłających do standardowego wyjścia (np. fprint, putchar)
    - zamiast przykładowej komendy : ./a.out używamy : ./a.out > anti.ppm

rastrowy

dostęp sekwencyjny



Przykład użycia tej techniki, sekwencyjny dostęp do danych:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <complex.h>// floor
/*
based on
c++ program from :
http://commons.wikimedia.org/wiki/File:Color_complex_plot.jpg
by      Claudio Rocchini
gcc d.c -lm -Wall
http://en.wikipedia.org/wiki/Domain_coloring
*/

const double PI = 3.14159265358979323846264338327950;
const double E  = 2.71828182845904523536028747135266;

/*
complex domain coloring
Given a complex number z=re^{ i \theta},
```

```
hue represents the argument ( phase, theta ),
sat and value represents the modulus
*/
int GiveHSV( double complex z, double HSVcolor[3] )
{
  //The HSV, or HSB, model describes colors in terms of hue, saturation, and value (brightness).

  // hue = f(argument(z))
  //hue values range from .. to ..
  double a = carg(z); //
  while(a<0) a += 2*PI; a /= 2*PI;
  // radius of z
  double m = cabs(z); //
  double ranges = 0;
  double rangee = 1;
  while(m>rangee){
      ranges = rangee;
      rangee *= E;
           }
  double k = (m-ranges)/(rangee-ranges);
  // saturation = g(abs(z))
  double sat = k<0.5 ? k*2: 1 - (k-0.5)*2;
  sat = 1 - pow( (1-sat), 3);
  sat = 0.4 + sat*0.6;
  // value = h(abs(z))
  double val = k<0.5 ? k*2: 1 - (k-0.5)*2;
      val = 1 - val;
      val = 1 - pow( (1-val), 3);
      val = 0.6 + val*0.4;

  HSVcolor[0]= a;
  HSVcolor[1]= sat;
  HSVcolor[2]= val;
return 0;
}


int GiveRGBfromHSV( double HSVcolor[3], unsigned char RGBcolor[3] ) {
                double r,g,b;
                double h; double s; double v;
                h=HSVcolor[0]; // hue
                s=HSVcolor[1]; //  saturation;
                v = HSVcolor[2]; // = value;
                if(s==0)
                                r = g = b = v;
                else {
                                if(h==1) h = 0;
                                double z = floor(h*6);
                                int i = (int)z;
                                double f = (h*6 - z);
                                double p = v*(1-s);
                                double q = v*(1-s*f);
                                double t = v*(1-s*(1-f));
                                switch(i){
                                                case 0: r=v; g=t; b=p; break;
                                                case 1: r=q; g=v; b=p; break;
                                                case 2: r=p; g=v; b=t; break;
                                                case 3: r=p; g=q; b=v; break;
                                                case 4: r=t; g=p; b=v; break;
                                                case 5: r=v; g=p; b=q; break;
                                }
                }
                int c;
                c = (int)(256*r); if(c>255) c = 255; RGBcolor[0] = c;
                c = (int)(256*g); if(c>255) c = 255; RGBcolor[1] = c;
```

```
            c = (int)(256*b); if(c>255) c = 255; RGBcolor[2] = c;
    return 0;
}
int GiveRGBColor( double complex z, unsigned char RGBcolor[3])
{
    static double HSVcolor[3];
    GiveHSV( z, HSVcolor );
    GiveRGBfromHSV(HSVcolor,RGBcolor);
    return 0;
}
//
double complex fun(double complex c ){
    return (cpow(c,2)-1)*cpow(c-2.0- I,2)/(cpow(c,2)+2+2*I);} //

int main(){
            // screen (integer ) coordinate
            const int dimx = 800; const int dimy = 800;
            // world ( double) coordinate
            const double reMin = -2; const double reMax =  2;
            const double imMin = -2; const double imMax =  2;

            static unsigned char RGBcolor[3];
            FILE * fp;
            char *filename ="complex.ppm";
            fp = fopen(filename,"wb");
            fprintf(fp,"P6\n%d %d\n255\n",dimx,dimy);

            int i,j;
            for(j=0;j<dimy;++j){
                        double im = imMax - (imMax-imMin)*j/(dimy-1);
                        for(i=0;i<dimx;++i){
                                    double re = reMax - (reMax-reMin)*i/(dimx-1);
                                    double complex z= re + im*I; //
                                    double complex v = fun(z); //
                                    GiveRGBColor( v, RGBcolor);

                                    fwrite(RGBcolor,1,3,fp);
                        }
            }
            fclose(fp);
            printf("OK - file %s saved\n", filename);
            return 0;
}
```

Przykład użycia tej techniki, swobodny dostęp do danych:

```
#include <stdio.h>
#include <stdlib.h> /* for ISO C Random Number Functions */
#include <math.h>
/*  gives sign of number */
double sign(double d)
{
          if (d<0)
          {return -1.0;}
          else {return 1.0;};
};
/* ----------------------*/
int main()
{
     const double Cx=0.0,Cy=1.0;
       /* screen coordinate = coordinate of pixels */
     int iX, iY,
           iXmin=0, iXmax=2000,
           iYmin=0, iYmax=2000,
           iWidth=iXmax-iXmin+1,
           iHeight=iYmax-iYmin+1,
           /* number of bytes = number of pixels of image * number of bytes of color */
           iLength=iWidth*iHeight*3,/* 3 bytes of color  */
           index; /* of array */
   int iXinc, iYinc;
   /* world ( double) coordinate = parameter plane*/
           const double ZxMin=-2.5;
           const double ZxMax=2.5;
           const double ZyMin=-2.5;
           const double ZyMax=2.5;
           /* */
           double PixelWidth=(ZxMax-ZxMin)/iWidth;
           double PixelHeight=(ZyMax-ZyMin)/iHeight;
           double Zx, Zy,    /* Z=Zx+Zy*i   */
                       NewZx, NewZy,
                       DeltaX, DeltaY,
                       SqrtDeltaX, SqrtDeltaY,
                       AlphaX, AlphaY,
                       BetaX,BetaY; /* repelling fixed point Beta */
             /*  */
           int Iteration,
                 IterationMax=100000000;
       /* PPM file */
     FILE * fp;
     char *filename="julia1.ppm";
     char *comment="# this is julia set for c=i ";/* comment should start with # */
     const int MaxColorComponentValue=255;/* color component ( R or G or B) is coded from 0 to 255 */
       /* dynamic 1D array for 24-bit color values */
     unsigned char *array;
 /*  ---------  find repelling fixed point ------------------------------*/
   /* Delta=1-4*c */
   DeltaX=1-4*Cx;
   DeltaY=-4*Cy;
   /* SqrtDelta = sqrt(Delta) */
   /* sqrt of complex number algorithm from Peitgen, Jurgens, Saupe: Fractals for the classroom */
   if (DeltaX>0)
   {
     SqrtDeltaX=sqrt((DeltaX+sqrt(DeltaX*DeltaX+DeltaY*DeltaY))/2);
     SqrtDeltaY=DeltaY/(2*SqrtDeltaX);
     }
     else /* DeltaX <= 0 */
     {
```

```
              if (DeltaX<0)
              {
                SqrtDeltaY=sign(DeltaY)*sqrt((-DeltaX+sqrt(DeltaX*DeltaX+DeltaY*DeltaY))/2);
                SqrtDeltaX=DeltaY/(2*SqrtDeltaY);
                }
                else /* DeltaX=0 */
                {
                  SqrtDeltaX=sqrt(fabs(DeltaY)/2);
                  if (SqrtDeltaX>0) SqrtDeltaY=DeltaY/(2*SqrtDeltaX);
                                    else SqrtDeltaY=0;
                        }
      };
    /* Beta=(1-sqrt(delta))/2 */
    BetaX=0.5+SqrtDeltaX/2;
    BetaY=SqrtDeltaY/2;
    /* Alpha=(1+sqrt(delta))/2 */
    AlphaX=0.5-SqrtDeltaX/2;
    AlphaY=-SqrtDeltaY/2;
    printf(" Cx= %f\n",Cx);
    printf(" Cy= %f\n",Cy);
    printf(" Beta= %f\n",BetaX);
    printf(" BetaY= %f\n",BetaY);
    printf(" AlphaX= %f\n",AlphaX);
    printf(" AlphaY= %f\n",AlphaY);
    /* initial value of orbit Z=Z0 is repelling fixed point */
    Zy=BetaY; /*  */
    Zx=BetaX;
    /*---------------------------------------------------------------*/
      array = malloc( iLength * sizeof(unsigned char) );
      if (array == NULL)
      {
         fprintf(stderr,"Could not allocate memory");
         getchar();
         return 1;
      }
      else
      {
         /* fill the data array with white points */
         for(index=0;index<iLength-1;++index) array[index]=255;
    /* ---------------------------------------------------------------*/
  for (Iteration=0;Iteration<IterationMax;Iteration++)
                  {
                        /* Zn*Zn=Z(n+1)-c */
                        Zx=Zx-Cx;
                        Zy=Zy-Cy;
                        /* sqrt of complex number algorithm from Peitgen, Jurgens, Saupe: Fractals for the
classroom */
                        if (Zx>0)
                        {
                          NewZx=sqrt((Zx+sqrt(Zx*Zx+Zy*Zy))/2);
                          NewZy=Zy/(2*NewZx);
                          }
                          else /* ZX <= 0 */
                          {
                            if (Zx<0)
                                {
                                  NewZy=sign(Zy)*sqrt((-Zx+sqrt(Zx*Zx+Zy*Zy))/2);
                                  NewZx=Zy/(2*NewZy);
                                  }
                                  else /* Zx=0 */
                                  {
                                    NewZx=sqrt(fabs(Zy)/2);
                                    if (NewZx>0) NewZy=Zy/(2*NewZx);
                                          else NewZy=0;
```

```
                                }
                            };
                        if (rand()<(RAND_MAX/2))
                        {
                                Zx=NewZx;
                                Zy=NewZy;
                                }
                        else {Zx=-NewZx;
                                Zy=-NewZy; }
                        /* translate from world to screen coordinate */
                        iX=(Zx-ZxMin)/PixelWidth;
                        iY=(Zy-ZyMin)/PixelHeight; /*  */
                        /* plot  pixel  */
                        array[((iYmax-iY-1)*iXmax+iX)*3]=0;
                        array[((iYmax-iY-1)*iXmax+iX)*3+1]=0;
                        array[((iYmax-iY-1)*iXmax+iX)*3+2]=0;
                    };
            /* write the whole data array to ppm file in one step */
            /*create new file,give it a name and open it in binary mode  */
            fp= fopen(filename,"wb"); /* b -  binary mode */
            if (fp == NULL){ fprintf(stderr,"file error"); }
                        else
                        {
                        /*write ASCII header to the file*/
                        fprintf(fp,"P6\n %s\n %d\n %d\n %d\n",comment,iXmax,iYmax,MaxColorComponentValue);
                        /*write image data bytes to the file*/
                        fwrite(array,iLength ,1,fp);
                        fclose(fp);
                        fprintf(stderr,"file saved");
                        getchar();
                        }
            free(array);
            return 0;
        } /* if (array ..  else ... */
}
```