**Đại học Bách khoa Hà Nội**

**Trường Công nghệ thông tin và truyền thông**

**----- o0o -----**

****

**BÁO CÁO PROJECT**

**Môn: Thực hành kiến trúc máy tính**

**Học phần: IT3280**

**Mã lớp: 130997**

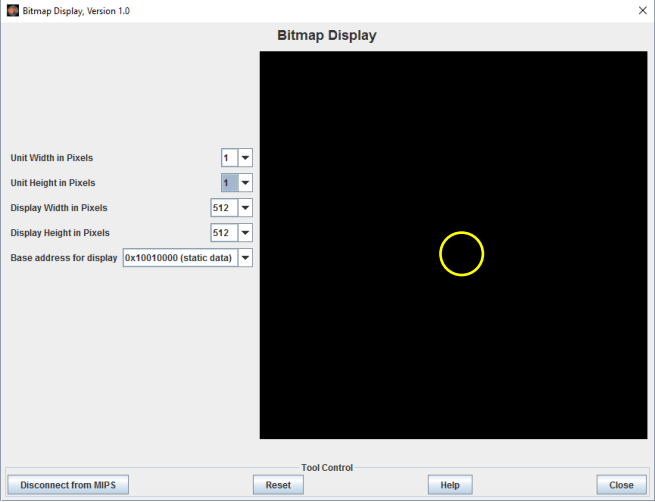
**Giảng viên hướng dẫn: Nguyễn Thị Thanh Nga**

**Nhóm sinh viên thực hiện:**

**+ Lê Thế Anh MSSV: 20200018**

**+ Phan Thế Anh MSSV: 20204941**

1. Project 2 – Lê Thế Anh\_20200018
2. Đề bài: Vẽ hình trên màn hình BitmapViết một chương trình sử dụng MIPS để vẽ một quả bóng di chuyển trên màn hình mô phỏng Bitmap của Mars). Nếu đối tượng đập vào cạnh của màn hình thì sẽ di chuyển theo chiều ngược lại.  
   Yêu cầu:  
   - Thiết lập màn hình ở kích thước 512x512. Kích thước pixel 1x1.  
   - Quả bóng là một đường tròn. Chiều di chuyển phụ thuộc vào phím người dùng bấm, gồm có (di chuyển lên (W), di chuyển xuống (S), Sang trái (A), Sang phải (D) trong bộ giả lập Keyboard and Display MMIO Simulator).Tốc độ bóng di chuyển là không đổi. Vị trí bóng ban đầu ở chính giữa màn hình



1. Phân tích bài toán

Bài toán yêu cầu vẽ một quả bóng di chuyển trên màn hình mô phỏng Bitmap của Mars. Nếu đối tượng đập vào cạnh của màn hình thì sẽ di chuyển theo chiều ngược lại. Việc xử lý bài toán sẽ được chia thành ba phần:

* Phần 1: Thiết lập màn hình ở kích thước 512x512. Kích thước pixel 1x1
* Phần 2: Vẽ bóng
* Phần 3: Di chuyển
* Phần 4: Phóng to, thu nhỏ

1. **Hướng giải quyết và các thuật toán liên quan**
2. Thiết lập màn hình **ở kích thước 512x512. Kích thước pixel 1x1**

Sử dụng bitmap display và connect với MIPS, đồng thời thiết lập màn hình ở kích thước 512x512, pixel 1x1

1. **Vẽ bóng**

* Để vẽ được bóng trên màn hình đầu tiên ta sẽ thiết lập thông số màu sắc và tọa độ điểm cho tâm của hình tròn ban đầu tại chính giữa màn hình ( tọa độ (256,256) ).
* Sử dụng tọa độ tâm của đường tròn để vẽ các pixel xung quanh bằng cách sử dụng thuật toán “midpoint circle algorithm” (thuật toán vòng tròn điểm giữa.)

Link chi tiết: <https://en.wikipedia.org/wiki/Midpoint_circle_algorithm>

1. Di chuyển

* Trước khi di chuyển ta sẽ phải thiết lập kí tự nhập vào trong bộ giả lập Keyboard and Display MMIO Simulatornhằm tạo hướng di chuyển
* Chiều di chuyển phụ thuộc vào phím người dùng bấm, gồm có

(di chuyển lên (w), di chuyển xuống (s), Sang trái (a), Sang phải (d) )

* Kiểm tra nảy:
* Qủa bóng sẽ nảy theo hướng ngược lại khi cạnh của nó chạm vào cạnh của màn hình. Để xác định vị trí nảy thì ta cần dùng tọa độ của tâm
* Khi quả bóng chạm cạnh màn hình thì quả bóng và rìa có ít nhất một điểm chung. Suy ra, khoảng cách từ tâm đến rìa ≤ bán kính.
* Kiểm tra nảy sẽ duyệt lần lượt toàn bộ theo 4 hướng: Trái, phải, trên, dưới
* Tiến hành nảy:

Khi điều kiện kiểm tra nảy thỏa mãn: ( Khoảng cách từ tâm hình tròn đến rìa màn hình ≤ bán kính ) thì sẽ thực hiện di chuyển (nảy) theo hướng ngược lại.

Để nảy theo hướng ngược lại cần đáp ứng các điều kiện:

* Người dùng chưa nhập hướng di chuyển mới hoặc người dùng nhập hướng di chuyển mới trùng với hướng di chuyển cũ.
* Nghĩa là quả bóng tiếp tục di chuyển theo hướng trước đó.
* Để làm cho quả bóng di chuyển thì ta sẽ tô màu đen cho quả bóng cũ và tô màu vàng cho quả bóng mới

1. Phóng to, thu nhỏ hình tròn

* Để phóng to hoặc thu nhỏ đường tròn rất đơn giản, ta chỉ cần thay đổi giá trị của bán kính tăng, giảm theo kí tự nhập vào
* Ví dụ: khi nhập vào kí tự ‘ + ‘ thì sẽ tăng bán kính lên 1 pixel, khi nhập kí tự ‘ - ’ thì sẽ giảm bán kính đi 1 pixel

1. Chi tiết về thuật toán midpoint circle algorithm và vẽ cách vẽ vòng tròn bát phân trong thuật toán

Ảnh có chứa văn bản, cửa shoji, tòa nhà

Mô tả được tạo tự động

Rasterizing a circle of radius 23 with the Bresenham midpoint circle algorithm. Only the green   octant is actually calculated, it's simply mirrored eight times to form the other seven octants.

* Mục tiêu của thuật toán là vẽ gần đúng đường tròn sử dụng pixel. ở mỗi lần lặp, đường tròn được hoàn thiện bằng cách chọn các pixel liền kề nhau thỏa mãn và các pixel tối đa
* Để thực hiện được thuật toán, đầu tiền cần xác định sai số bán kính RE có tâm tại (
* Xét tọa độ ban đầu của tâm tại (r, 0) =>
* Vì vòng tròn được vẽ bắt đầu từ góc phần tư thứ nhất nên ta sẽ lần lượt tăng y lên 1 đơn vị sau mỗi lần lặp
* Mặt khác vì được vẽ theo vòng tròn bát phân nên giá trị của x chỉ có thể giữ nguyên hoặc giảm đi 1. Xét bất đẳng thức sau

⬄ <

⬄ <

* Bình phương hai vế:

⬄

* Vì x>0 và nên

(

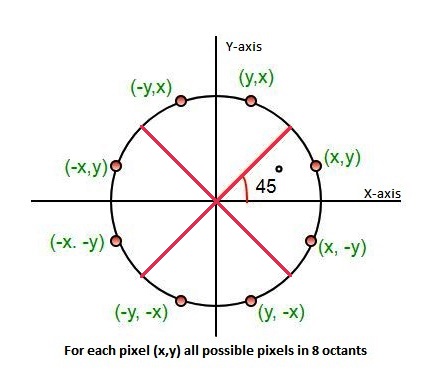
⬄ 2[

Nếu 2[ thì giảm giá trị X xuống 1 đơn vị

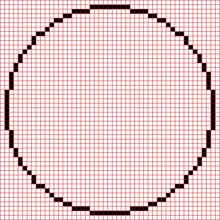
Nếu 2[ thì giữ nguyên giá trị X

Vẽ vòng tròn bát phân

Sau khi xác định được các giá trị x,y ta sẽ vẽ chúng trong vòng tròn bát phân bằng cách vẽ các tọa độ: [+x,+y]; [+y,+x]; [-x,+y]; [-y,+x];[-x,-y]; [-y,-x]; [+x,-y]; [+y,-x]



Thực hiện vòng lặp cho đến khi điều kiện x < y thỏa mãn ta sẽ được đường tròn gần như là hoàn hảo



A circle of radius 23 drawn by the Bresenham algorithm

1. Mã giả

radius = 10 ; // bán kính

initial\_x = 256 ; // Tọa độ tâm x ban đầu

initial\_y = 256 ; // Tọa độ tâm y ban đầu

char old\_dir, new\_dir ; // hướng cũ, hướng mới

int cx, cy, x, y; //tâm (cx,cy), điểm (x,y)

void load\_key()

{

Nhập: new\_dir;

}

// di chuyển

void move(){

while(1)

{

new\_dir = 0 ;

load\_key();

switch(new\_dir)

{

case ‘+’: check\_bonuce() -> ZoomOut -> break ;

case ‘-’: check\_bonuce() -> ZoomIn -> break ;

case 'a': check\_bonuce() -> move\_left() -> break;

case 'd': check\_bonuce() -> move\_right() -> break;

case 'w': check\_bonuce() -> move\_up() -> break;

case 's': check\_bonuce() -> move\_down() -> break;

default:

switch(old\_dir)

{

case 'a': check\_bonuce() -> move\_left() -> break;

case 'd': check\_bonuce() -> move\_right() -> break;

case 'w': check\_bonuce() -> move\_up() -> break;

case 's': check\_bonuce() -> move\_down() -> break;

}

break;

}

}

}

void check\_bonuce() // kiểm tra nảy

{

if(cx <= 13){

if( ((old\_dir == 'a') and (new\_dir == 0)) or (new\_dir == 'a')) move\_right();

}

if(cx >= 499){

if( ((old\_dir == 'd') and (new\_dir == 0)) or (new\_dir == 'd')) move\_left();

}

if(cy <= 13){

if( ((old\_dir == 'w') and (new\_dir == 0)) or (new\_dir == 'w')) move\_down();

}

if(cy >= 499)

{

if( ((old\_dir == 's') and (new\_dir == 0)) or (new\_dir == 's')) move\_up();

}

}

// thu nhỏ

void ZoomIn()

{

color = black;

draw\_circle(cx,cy,x,y) ;

old\_dir = '-' ;

radius = radius -1 ;

color = yellow ;

draw\_circle (cx,cy,x,y) ;

}

// Phóng to

void ZoomOut()

{

color = black;

draw\_circle(cx,cy,x,y) ;

old\_dir = '+' ;

radius = radius +1 ;

color = yellow ;

draw\_circle (cx,cy,x,y) ;

}

// Rẽ trái

void move\_left()

{

color = black;

draw\_circle(cx,cy,x,y) ;

old\_dir = 'a' ;

cx = cx -2 ;

color = yellow ;

draw\_circle (cx,cy,x,y) ;

}

// Rẽ phải

void move\_right() ; // Tương tự

//Rẽ lên

void move\_up() ;

// Rẽ xuống

void move\_down() ;

// Vẽ hình tròn với tọa độ tâm (cx,cy); tọa độ điểm (x,y) bằng thuật toán midpoint circle algorithm

void draw\_circle(cx,cy,x,y)

{

x = radius;

y = 0;

Xchg = 1-2\*x;

Ychg = 1 ;

Re = 0 ;

while (x >= y)

{

set\_pixel(cx + x, cy + y);

set\_pixel(cy + y, cx + x);

set\_pixel(cx - x, cy + y);

set\_pixel(cy - y, cx + x);

set\_pixel(cx - x, cy - y);

set\_pixel(cy - y, cx - x);

set\_pixel(cx + x, cy - y);

set\_pixel(cy + y, cx - x);

y++;

Re += Ychg;

Ychg = Ychg +2 ;

if ((2\*Re + Xchg) > 0)

x = x-1 ;

Re += Xchg;

Xchg = Xchg + 2;

}

}

int main(){

cx = intinital\_x ;

cy = intinital\_y ;

x = radius;

y = 0;

Xchg = 1-2\*x;

Ychg = 1 ;

Re = 0 ;

draw\_circle(cx,cy,x,y);

move();

}

1. Ý nghĩa các thanh ghi và các chương trình con
2. Ý nghĩa các thanh ghi

* Thanh ghi $a0, $a1: chứa giá trị đầu vào của lời gọi hàm, đồng thời lưu giá trị tọa độ tâm (x , y) của đường tròn và tọa độ điểm

( x, y )

* Thanh ghi $a2 : lưu giá trị màu sắc
* Thanh ghi $a3: lưu giá trị bán kính hình tròn
* Thanh ghi $k0: chứa kí tự nhập vào
* Thanh ghi $k1: kiểm tra xem đã nhập phím nào chưa
* Thanh ghi $s0, $s1: lưu giá trị của tọa độ x và y trong hàm draw\_circle
* Thanh ghi $s2: Lưu giá trị của biến raderr trong hàm draw\_circle
* Thanh ghi $s4: Lưu giá trị của biến Ychg trong hàm draw\_circle
* Thanh ghi $s3: Lưu giá trị hướng đi mới, đồng thời chứa giá trị của biến Xchg trong hàm draw\_circle
* Thanh ghi $s5: Lưu giá trị hướng đi cũ
* Thanh ghi $t8, $t9: Lưu giá trị tọa độ tâm (x,y) trong hàm draw8
* Thanh ghi $t3, $t4, $t5: Chứa giá trị của các hàm boollen 0 hoặc 1.
* Thanh ghi $v0: Chứa giá trị trả về của lời gọi hàm
* Thanh ghi $ra: Lưu địa chỉ của lệnh tiếp theo sau chương trình con

1. Chương trình con

* draw8:
* subprogram: draw8
* Purpose: draw single point in all 8 octants
* Input: arguments:

$s0 -- ( tọa độ x)

$s1 -- ( tọa dộ y)

registers:

$t8 -- ( tâm x )

$t9 -- ( tâm y )

* Output: pixel of (x,y)
* setpixel:
* Subprogram: setpixel
* Purpose: draw pixel on display
* Input: arguments:
  + - $a0 – tọa độ x
    - $a1 – tọa độ y

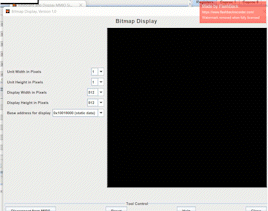
clobbers:

$v0 -- bitmap offset/index

$v1 -- bitmap address

* Output: pixel on display
* draw\_circle:
* Subprogram: draw\_circle
* Purpose: draw circle with midpoint circle algorithm
* Input:
  + - $s0: coord x
    - $s1: coord y
    - $s3: Xchange in midpoint circle algorithm
    - $s2: Raderr in midpoint circle algorithm
    - $s4: Ychange in midpoint circle algorithm
* Output: circle is drawed by midpoint circle algorithm

1. Hình ảnh,kết quả mô phỏng



1. Cách sử dụng

* Bước 1: mở file LeTheAnh\_20200018\_Project2.asm bằng chương trình MARS
* Bước 2: Bật BITMAP DISPLAY và Keyboard and Display MMIO Simulator trong phần tools và connect với MIPS
* Bước 3: thiết lập màn hình bitmap display ở kích thước 512x512, và 1x1 pixel
* Bước 4: Nhập từ khóa điều khiển hướng đi của bóng trong Display MMIO Simulator và tiến hành chạy chương trình.

1. Project 8 – Phan Thế Anh\_20204941
2. Đề bài: Mô phỏng ổ đĩa RAID 5

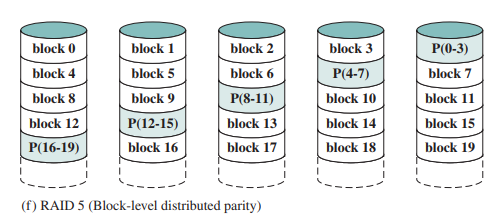
* Hệ thống ổ đĩa RAID5 cần tối thiểu 3 ổ đĩa cứng, trong đó phần dữ liệu parity sẽ được chứa lần lượt lên 3 ổ đĩa như trong hình bên. Hãy viết chương trình mô phỏng hoạt động của RAID 5 với 3 ổ đĩa, với giả định rằng, mỗi block dữ liệu có 4 kí tự. Giao diện như trong minh họa dưới. Giới hạn chuỗi kí tự nhập vào có độ dài là bội của 8.
* Trong ví dụ sau, chuỗi kí tự nhập vào từ bàn phím (DCE.\*\*\*\*ABCD1234HUSTHUST) sẽ được chia thành các block 4 byte. Block 4 byte đầu tiên “DCE.” sẽ được lưu trên Disk 1, Block 4 byte tiếp theo “\*\*\*\*” sẽ lưu trên Disk 2, dữ liệu trên Disk 3 sẽ là 4 byte parity được tính từ 2 block đầu tiên với mã ASCII là 6e=’D’ xor ‘\*’ ; 69=’C’ xor ‘\*’; 6f=’E’ xor ‘\*’ ; 04=’.’ xor ‘\*’

Ảnh có chứa văn bản

Mô tả được tạo tự động

1. Phân tích bài toán

* RAID là chữ viết tắt của Redundant Array of Independent Disks. Ban đầu, RAID được sử dụng như một giải pháp phòng hộ vì nó cho phép ghi dữ liệu lên nhiều đĩa cứng cùng lúc. Về sau, RAID đã có nhiều biến thể cho phép không chỉ đảm bảo an toàn dữ liệu mà còn giúp gia tăng đáng kể tốc độ truy xuất dữ liệu từ đĩa cứng.
* RAID có dung lượng lớn, truy cập song song và an toàn thông tin
* RAID 5 có thể tách ra lưu trữ các ổ cứng riêng biệt và vẫn có phương án dự phòng khi có sự cố phát sinh đối với 1 ổ cứng bất kỳ trong cụm. Đây là dạng RAID mạnh mẽ, được sử dụng phổ biến nhất hiện nay.
* Phần dữ liệu parity sẽ được tính toán dựa trên dữ liệu từ 2 ổ còn lại bằng cách xor 4 byte dữ liệu từ 2 ổ trên với nhaau để tạo ra 4 byte parity.



* Như vậy, trong bài toán này, ta cần phải tìm cách phân phối từng block dữ liệu sao cho hợp lý, đúng với quy tắc. Nghĩa là dữ liệu parity sẽ được lưu trữ trên ổ theo quy luật như hình trên.

1. Cách làm

* Trước tiên, ta cần phải kiểm tra độ dài chuỗi kí tự mà người dùng nhập vào. Giới hạn chuỗi kí tự có độ dài là bội của 8
* Tiếp theo, ta sẽ xử lý lần lượt 8 kí tự cho mỗi lần ghi:
  + 4 kí tự đầu tiên sẽ được lưu trữ vào mảng 1
  + 4 kí tự cuối cùng sẽ được lưu trữ vào mảng 2
  + 4 byte parity sẽ được tính bằng cách lấy từng kí tự của mảng 1 và mảng 2 XOR với nhau.
* Sau khi phân chia dữ liệu, ta sẽ thực hiện việc sắp xếp dữ liệu đó vào các ổ đĩa. Để việc sắp xếp trở lên chính xác, ta sẽ sử dụng một biến điểm cout với giá trị khởi tạo là 1. Ta lấy phần dư từ phép cout chia 3, kết quả là:
  + Dư = 1: mảng 1 lưu vào Disk 1, mảng 2 lưu vào Disk 2, dữ liệu parity lưu vào Disk 3
  + Dư = 2: mảng 1 lưu vào Disk 1, mảng 2 lưu vào Disk 3, dữ liệu parity lưu vào Disk 3
  + Dư = 0: mảng 1 lưu vào Disk 2, mảng 2 lưu vào Disk 3, dữ liệu parity lưu vào Disk 1
* Vòng lặp kết thúc khi ta xét đến 8 kí tự cuối cùng của chuỗi kí tự

1. Mã giả

# String: Chuỗi kí tự người dùng nhập vào

# Kiểm tra độ dài chuỗi kí tự

i = 0;

c=string[i];

Cout=0;

while ( c != “\n”)

{

Cout++;

i++;

c=string[i];

}

if ( c mode 8 !=)

return 0;

# Thực hiện vòng lặp lưu dữ liệu vào các ổ

Cout = Cout/8;

K = 0;

Index = 0;

while (K < Cout)

{

K++;

# Vòng lặp để lưu trữ kí tự vào mảng

for ( i=0; i<7; i++)

{

if ( i<4 )

mang1[i] = string[Index];

else mang2[i-4] = string[Index];

Index++;

}

# Sắp xếp dữ liệu vào ổ và tính toán dữ liệu parity

Check = K mode 3;

if ( Check = 1 )

printf(mang1, mang2, parity);

else if (Check = 2)

Printf(mang1, parity, mang2);

else printf(“parity,mang1,mang2);

printf(“\n”);

}

# Hàm tính toán dữ liệu parity

void parity(mang1[], mang2[])

{

for (i=0; i<4; i++)

{

mang3[i] = mang1[i] XOR mang2[i];

# In ra giá trị 4 bit gần cuối theo mã hexa

X = Mang3[i] dịch phải4 bit;

X = X AND 0xf;

printf(X);

# In ra giá trị 4 bit cuối

X = Mang3[i] AND 0xf;

printf(X);

if ( i < 3 )

printf(“,”);

}

}

1. Chương trình con

|  |  |  |  |
| --- | --- | --- | --- |
| **Tên** | **Input** | **Output** | **Chức năng** |
| PromptString | $a0 – chuỗi thông báo  $a1 – kích thước của chuỗi | $v0 – địa chỉ của chuỗi người dùng nhập vào | Nhập và cung cấp bộ nhớ động cho chuỗi người dùng nhập vào |
| PrintString | $a0 – địa chỉ chuỗi cần in | None | In ra màn hình chuỗi kí tự cần hiển thị |
| PromptInt | $a0 – địa chỉ chuỗi thông báo yêu cầu người dùng nhập số nguyên | $v0 – số nguyên người dùng nhập vào | Lấy được một giá trị là số nguyên từ người dùng |
| Print\_string\_array | $a0 - địa chỉ chuỗi | none | In ra mảng chuỗi kí tự |
| Write\_Disk3 | $s1 - address of Disk 1  $s2 - address of Disk 2 | none | In ra màn hình 2 số cuối của giá trị hexa |
| PrintNewLine | none | none | In ra một dòng mới |
| Exit | None | None | Thoát chương trình |

1. Chức năng khôi phục dữ liệu
2. Ý tưởng

Thực hiện phép xor giữa 4 byte parity với 4 byte dữ liệu của ổ chưa bị mất dự liễu.

1. Cách thực hiện

* Bước 1: Hỏi người dùng có muốn sử dụng chức năng hay không. Nếu có sang bước 2, nếu không thì kết thúc.
* Bước 2: Yêu cầu người dùng nhập vào 8 ký tự
* Bước 3: Sắp xếp dữ liệu và in ra màn hình
* Bước 4: Hỏi người dùng muốn khôi phục dữ liệu nào
* Bước 5: Thực hiện phép xor giữa parity và ổ đĩa còn dữ liệu và in ra màn hình

1. Mã giả

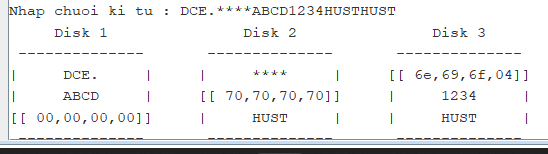
for i=0:3

printf(“%c”, Disk1[i] XOR Disk3[i]);

1. Kết quả mô phỏng

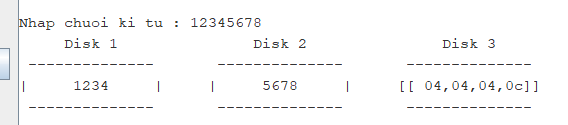
\* Cách sử dụng chương trình:

* Bước 1: Nhập chuỗi kí tự
* Bước 2: Sau khi có kết quả, chương trình sẽ thông báo về việc sử dụng chương trình demo khôi phục dữ liệu. Nếu có sang bước 3, nếu không sang bước 6
* Bước 3: Nhập chuỗi kí tự có độ dài là 8
* Bước 4: Nhập ổ đĩa cần khôi phục ( ổ 1 hoặc 2), nếu sai chương trình sẽ bắt nhập lại)
* Bước 5: Sau khi có kết quả khôi phục thì chương trình sẽ thông báo về việc sủ dụng lại chương trình. Nếu có quay lại bước 1, nếu không thì sang bước 6
* Bước 6: Ghi ra lời cảm ơn và kết thúc chương trình



Ảnh có chứa bàn

Mô tả được tạo tự động



Ảnh có chứa văn bản

Mô tả được tạo tự động