TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN – ĐHQG-HCM
KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG

BÁO CÁO THỰC HÀNH

Bài thực hành số 05: Tìm hiểu cơ bản về Kỹ thuật Dịch ngược

Môn học: Lập trình hệ thống

Lớp: NT209.Q13.ANTT.2

THÀNH VIÊN THỰC HIỆN (Nhóm 6):

STT	Họ và tên	MSSV
1	Huỳnh Đăng Khoa	24520815
2	Phan Hoàng Dũng	24520348
3	Phạm Ngọc Dũng	24520346

NỘI DUNG BÁO CÁO:

1. Pha 1

- Mã Assembly:

```
.text:0804898B                push    ebp
.text:0804898C                mov     ebp, esp
.text:0804898E                sub     esp, 38h
.text:08048991                lea     eax, [ebp+var_2C]
.text:08048994                add     eax, 14h
.text:08048997                push    eax
.text:08048998                lea     eax, [ebp+var_2C]
.text:0804899B                add     eax, 10h
.text:0804899E                push    eax
.text:0804899F                lea     eax, [ebp+var_2C]
.text:080489A2                add     eax, 0Ch
.text:080489A5                push    eax
.text:080489A6                lea     eax, [ebp+var_2C]
.text:080489A9                add     eax, 8
.text:080489AC                push    eax
.text:080489AD                lea     eax, [ebp+var_2C]
.text:080489B0                add     eax, 4
.text:080489B3                push    eax
.text:080489B4                lea     eax, [ebp+var_2C]
.text:080489B7                push    eax
.text:080489B8                push    offset aDDDDDD  ; "%d %d %d %d %d %d"
.text:080489BD                push    [ebp+arg_0]
.text:080489C0                call    ___isoc99_sscanf
.text:080489C5                add     esp, 20h
.text:080489C8                mov     [ebp+var_10], eax
.text:080489CB                cmp     [ebp+var_10], 6
.text:080489CF                jz      short loc_80489D6
```

```
.text:080489CF                  jz      short loc_80489D6
.text:080489D1                  call    explode_bomb
```

- Mã giả:

```
int __cdecl phase1(int a1)
{
  int result; // eax@4
  int v2; // [sp+Ch] [bp-2Ch]@1
  int v3; // [sp+10h] [bp-28h]@1
  int v4; // [sp+14h] [bp-24h]@1
  int v5; // [sp+18h] [bp-20h]@1
  int v6; // [sp+1Ch] [bp-1Ch]@1
  int v7; // [sp+20h] [bp-18h]@1
  int v8; // [sp+24h] [bp-14h]@3
  int v9; // [sp+28h] [bp-10h]@1
  int i; // [sp+2Ch] [bp-Ch]@6

  v9 = __isoc99_sscanf(a1, "%d %d %d %d %d %d", &v2, &v3, &v4, &v5, &v6, &v7);
  if ( v9 != 6 )
    explode_bomb();
  v8 = 1;
  if ( v2 || (result = v3, v3 != v8) )
    explode_bomb();
  for ( i = 2; i <= 5; ++i )
  {
    result = *(&v2 + i);
    if ( result != *(&v2 + i - 2) + *(&v2 + i - 1) )
      explode_bomb();
  }
  return result;
}
```

- **Định dạng input:** là 6 số nguyên (v2,v3,v4,v5,v6,v7).

- Điều kiện ràng buộc input:

+ Đầu vào của pha 1 là a1 được gán vào v9 và tiếp đó là if so sánh v9 với 6. Nếu không bằng nhau thì sẽ gọi hàm explode_bomb() nên input phải có 6 số nguyên.

+ Vòng lặp for tính giá trị result với i được khởi tạo = 2 và i <= 5 ;sau mỗi lần tính kiểm tra giá trị result với tổng của (v2+i-2) và (v2+i-1).Ta nhận ra rằng đây là công thức tính số Fibonacci có công thức tổng quát là Fn = Fn-1 + Fn-2 với n là (v2+i) → dãy số cần nhập là 1 dãy số Fibonacci.

- Kết luận về input:

+ Input cần nhập phải có 6 số nguyên.

+ Input phải là dãy số Fibonacci.

- Cách tìm được đáp án nộp trên vlab:

- Đầu tiên pha 1, ta nhìn thấy đầu vào của pha 1 là a1 dạng int và đưa số này vào v9. Sau đó, ta lại thấy nếu v9 != 6 thì bom nổ (gọi hàm explode_bomb()) → ràng buộc v9 phải có 6 số nên input cần nhập có 6 số nguyên.

- Tiếp đó, ta biết được giá trị v8 = 1. Sau đó lại có ràng buộc:

```
if ( v2 || (result = v3, v3 != v8) )
  explode_bomb();
```

- Điều kiện của if trên là v2 or với đáp án của điều kiện v3 != v8 . Nếu thỏa if trên thì sẽ gọi hàm nổ bom → if trên phải sai ở cả 2 vế nên ta có thể suy ra rằng v2 = 0 và v3 = 1 (vì v8 đã bằng 1) → đáp án đã có 2 kí tự đầu là 0 và 1.

- Sau đó, ta lại có vòng lặp:

```
for ( i = 2; i <= 5; ++i )
{
  result = *(&v2 + i);
  if ( result != *(&v2 + i - 2) + *(&v2 + i - 1) )
    explode_bomb();
}
return result;
```

- Với mỗi i = 2 và i <= 5, ta tính được giá trị result = v2 + 1, tiếp đó là điều kiện if so sánh result với tổng 2 tổ hợp là (v2 + i - 2) và (v2 + i - 1). Đây chính là công thức tính số Fibonacci quen thuộc với công thức tổng quát Fn = Fn-1 + Fn-2 với n là (v2 + i). Vậy nếu result ở trên khác với công thức tính số Fibonacci thì sẽ gọi hàm nổ bom → Đáp án là 1 dãy số Fibonacci.

Gía trị i	Gía trị result
2	1
3	2
4	3
5	5

→ Vậy input cần nhập dần lộ ra là 1 chuỗi số Fibonacci với 6 số và 2 số đầu tiên là 0 và 1. Đáp án là 0 1 1 2 3 5.

```
khoa@khoa-VMware-Virtual-Platform:~$ cd Lab5
khoa@khoa-VMware-Virtual-Platform:~/Lab5$ ./nt209-uit-bomb
Welcome to UIT's bomb lab.
You have to deactivate our bomb by solving 6 phases with the correc
t inputs consecutively, and otherwise the bomb will be blown up!

[*] Phase 1
- Hint: Numbers are always magical!
0 1 1 2 3 5
Good job! You've cleared the first phase!
```

- Kiểm tra đáp án trên, ta xác nhận đây là đáp án đúng.

2. Pha 2

- Mã assembly:

```
.text:08048A2D                push    ebp
.text:08048A2E                mov     ebp, esp
.text:08048A30                sub     esp, 28h
.text:08048A33                mov     [ebp+var_C], 0Bh
.text:08048A3A                mov     eax, [ebp+var_C]
.text:08048A3D                mov     eax, QUESTIONS[eax*4]
.text:08048A44                mov     [ebp+var_10], eax
.text:08048A47                mov     eax, [ebp+var_C]
.text:08048A4A                mov     eax, QA_MAP[eax*4]
.text:08048A51                mov     [ebp+var_14], eax
.text:08048A54                mov     eax, [ebp+var_14]
.text:08048A57                mov     eax, ANSWERS[eax*4]
.text:08048A5E                mov     [ebp+s2], eax
.text:08048A61                push    [ebp+arg_0]
.text:08048A64                call    transfer
.text:08048A69                add     esp, 4
.text:08048A6C                mov     [ebp+s1], eax
.text:08048A6F                mov     eax, [ebp+s2]
.text:08048A72                movzx   eax, byte ptr [eax]
.text:08048A75                test    al, al
.text:08048A77                jz      short loc_8048A8E
.text:08048A79                sub     esp, 8
.text:08048A7C                push    [ebp+s2]        ; s2
.text:08048A7F                push    [ebp+s1]        ; s1
.text:08048A82                call    is_equal
.text:08048A87                add     esp, 10h
.text:08048A8A                test    eax, eax
```

```
.text:08048A8A                test    eax, eax
.text:08048A8C                jnz     short loc_8048A93
.text:08048A8E
.text:08048A8E loc_8048A8E:                           ; CODE XREF: phase2+4A↑j
.text:08048A8E                call    explode_bomb
```

- Mã giả:

```
1 char __cdecl phase2(int a1)
2 {
3   char *v1; // ST28_4@1
4   int v2; // eax@2
5   char *s1; // [sp+Ch] [bp-1Ch]@1
6   char *s2; // [sp+10h] [bp-18h]@1
7
8   v1 = QUESTIONS[11];
9   s2 = ANSWERS[*(&QA_MAP + 11)];
0   s1 = (char *)transfer(a1);
1   if ( !*s2 || (LOBYTE(v2) = is_equal(s1, s2), !v2) )
2     explode_bomb();
3   return v2;
4 }
```

- Định dạng input: 1 chuỗi kí tự (s1)

- Điều kiện ràng buộc input: Ta thấy s2 là lấy phần tử từ mảng answers và s1 là input được qua hàm transfer và gán kiểu chuỗi. Sau đó lại so sánh 2 chuỗi này, nếu không bằng sẽ kích hoạt hàm nổ bom → input phải bằng với đáp án của s2.

- Kết luận về input: input cần nhập phải bằng với s2.

- Cách tìm đáp án nộp trên vLab:

- Ban đầu pha 2, ta thấy đầu vào của pha là a1 dạng int, v1 là phần tử thứ 11 của mảng question. Ta tìm thử trong mảng question:

```
.data:0804C060                 public QUESTIONS
.data:0804C060 QUESTIONS       dd offset aWhatIsTheCapit ; DATA XREF: phase2+10↑r
.data:0804C060                                           ; "What is the capital of Thailand?"
.data:0804C064                 dd offset aWhatIsTheEngli ; "What is the English name of our course?"...
.data:0804C068                 dd offset aWhichSeasonHas ; "Which season has cherry blossoms?"
.data:0804C06C                 dd offset aWhichCountryIs ; "Which country is the Lion city in South"...
.data:0804C070                 dd offset aIf2024IsTheYea ; "If 2024 is the year of Dragon, what ani"...
.data:0804C074                 dd offset aWhichIsTheLarg ; "Which is the largest net in the world?"
.data:0804C078                 dd offset aWhatIsTheLarge ; "What is the largest country in the worl"...
.data:0804C07C                 dd offset aWhatWordIsSpel ; "What word is spelled incorrectly in eve"...
.data:0804C080                 dd offset aWhatIsYourNati ; "What is your nationality?"
.data:0804C084                 dd offset aWhatIsThePhone ; "What is the phone number of our univers"...
.data:0804C088                 dd offset aCompleteTheDom ; "Complete the domain of our faculty: htt"...
.data:0804C08C                 dd offset aWhatIsTheNameO ; "What is the name of the analyzed execut"...
.data:0804C090                 dd offset aWhichCityHas37 ; "Which city has 3/7 of a chicken and 2/3"...
.data:0804C094                 dd offset aInUitWh:aWhatIsTheNameO db 'What is the name of the analyzed executable file in this lab?',0
.data:0804C098                 dd offset aThanksTo                                    ; DATA XREF: .data:0804C08C↓o
.data:0804C09C                 dd offset aWhatIsTheLonge ; "What is the longest wall in the world? "...
.data:0804C0A0                 dd offset aWhatIsTheEstab ; "What is the establishment date of UIT ("...
.data:0804C0A4                 dd offset aWhatIsTheFullE ; "What is the full English name of our un"...
.data:0804C0A8                 dd offset aIAmAnOddNumber ; "I am an odd number. Take away one lette"...
.data:0804C0AC                 dd offset aWhichFastFoodB ; "Which fast food brand features a bee in"...
.data:0804C0B0                 dd offset aWhichContinent ; "Which continent has the least populatio"...
.data:0804C0B4                 dd offset aWhatAreEaxEbxE ; "What are eax, ebx, eip, esi on a comput"...
```

- Ta tìm thấy phần tử 11 của mảng question là 1 chuỗi câu hỏi là " What is the name of the analyzed executable file in this lab?" . Tiếp theo đó, ta thấy s1 là input a1 được chuyển thành dạng chuỗi ; s2 là phần tử thứ 11 của mảng QA_MAP và phần tử đó chứa trong mảng answer, tìm thử chỉ số index của QA_MAP:

```
.data:0804C0E0                 public QA_MAP
.data:0804C0E0 QA_MAP          dd 0                    ; DATA XREF: phase2+1D↑r
```

- Ta thấy chỉ số index của QA_MAP mang giá trị 0. Vậy s2 là phần tử thứ 11 của mảng answer. Ta tìm thử trong mảng answer:

```
data:0804C160                  public ANSWERS
data:0804C160 ANSWERS          dd offset aJivosws      ; DATA XREF: phase2+2A↑r
data:0804C160                                          ; "Jivosws"
data:0804C164                  dd offset aKwuxcbmzAgabmu ; "Kwuxcbmz Agabmu Xzwoziuuqvo"
data:0804C168                  dd offset aAxzqvo        ; "Axzqvo"
data:0804C16C                  dd offset aAqvoixwzm     ; "Aqvoixwzm"
data:0804C170                  dd offset aAvism         ; "Avism"
data:0804C174                  dd offset aQvbmzvmb      ; "Qvbmzvmb"
data:0804C178                  dd offset aZcaaqi        ; "Zcaaqi"
data:0804C17C                  dd offset aQvkwzzmkbtg   ; "Qvkwzzmkbtg"
data:0804C180                  dd offset aDqmbviumam    ; "Dqmbviumam"
data:0804C184                  dd offset a80615030880   ; "80615030880"
data:0804C188                  dd offset aVk_cqb_mlc_dv ; "vk.cqb.mlc.dv"
data:0804C18C                  dd offset aVb087CqbJwuj  ; "vb087-cqb-jwuj"
data:0804C190                  dd offset aKpqkiow       ; "Kpqkiow"
data:0804C194                  dd offset aIzbqn:aVb087CqbJwuj db 'vb087-cqb-jwuj',0  ; DATA XREF: .data:0804C18C↓o
data:0804C198                  dd offset aEqvlwe        ; "Eqvlwe"
```

- Ta tìm thấy đáp án với định dạng như trên và tiếp sau đó là hàm if:

```
if ( !*s2 || (LOBYTE(v2) = is_equal(s1, s2), !v2) )
  explode_bomb();
return v2;
```

- Hàm if là kết quả của việc kiểm tra s2 có rỗng hoặc không hợp lệ hay không và kết quả của hàm so sánh 2 chuỗi s1 và s2 (với s1 là input). Nếu thỏa điều kiện của if thì sẽ gọi hàm nổ bom nên if phải sai; if phải sai khi cả 2 vế đều sai là s2 không được là rỗng và kết quả của hàm so sánh chuỗi phải trả về là 1 (chuỗi bằng nhau).

- Kết hợp điều kiện của if trên với chuỗi câu hỏi ta tìm được ở v1 và định dạng của đáp án ở s2, ta có thể dễ dàng đoán được input cần nhập là tên của file cần được phân tích → đáp án là nt209-uit-bomb.

```
[*] Phase 2
- Hint: You must answer your secret question!
nt209-uit-bomb
Two phases have been solved. Keep going!
```

- Ta xác nhận đáp án trên, ta thấy đó là đáp án đúng.

3. Pha 3

- **Mã Assembly:**

```
                public phase3
phase3          proc near               ; CODE XREF: main+11E↓p

var_18          = dword ptr -18h
var_14          = dword ptr -14h
var_10          = dword ptr -10h
var_C           = dword ptr -0Ch
arg_0           = dword ptr  8

                push    ebp
                mov     ebp, esp
                sub     esp, 18h
                mov     [ebp+var_C], 0
                mov     [ebp+var_10], 0
                lea     eax, [ebp+var_18]
                push    eax
                lea     eax, [ebp+var_14]
                push    eax
                push    offset aDD      ; "%d %d"
                push    [ebp+arg_0]
                call    ___isoc99_sscanf
                add     esp, 10h
                mov     [ebp+var_10], eax
                cmp     [ebp+var_10], 1
                jg      short loc_8048AD0
                call    explode_bomb

; ---------------------------------------------------------------------------

loc_8048AD0:                            ; CODE XREF: phase3+33↑j
                mov     eax, [ebp+var_14]
                cmp     eax, 7          ; switch 8 cases
                ja      short loc_8048B09 ; jumptable 08048ADF default case
                mov     eax, ds:off_8049AE8[eax*4]
                jmp     eax             ; switch jump
; ---------------------------------------------------------------------------

loc_8048AE1:                            ; CODE XREF: phase3+49↑j
                                        ; DATA XREF: .rodata:off_8049AE8↓o
                sub     [ebp+var_C], 68h ; jumptable 08048ADF case 0

loc_8048AE5:                            ; CODE XREF: phase3+49↑j
                                        ; DATA XREF: .rodata:off_8049AE8↓o
                sub     [ebp+var_C], 263h ; jumptable 08048ADF case 1

loc_8048AEC:                            ; CODE XREF: phase3+49↑j
                                        ; DATA XREF: .rodata:off_8049AE8↓o
                add     [ebp+var_C], 38h ; jumptable 08048ADF case 2

loc_8048AF0:                            ; CODE XREF: phase3+49↑j
                                        ; DATA XREF: .rodata:off_8049AE8↓o
                sub     [ebp+var_C], 389h ; jumptable 08048ADF case 3
```

```
loc_8048AF7:                            ; CODE XREF: phase3+49↑j
                                        ; DATA XREF: .rodata:off_8049AE8↓o
                add     [ebp+var_C], 68h ; jumptable 08048ADF case 4

loc_8048AFB:                            ; CODE XREF: phase3+49↑j
                                        ; DATA XREF: .rodata:off_8049AE8↓o
                sub     [ebp+var_C], 68h ; jumptable 08048ADF case 5

loc_8048AFF:                            ; CODE XREF: phase3+49↑j
                                        ; DATA XREF: .rodata:off_8049AE8↓o
                add     [ebp+var_C], 68h ; jumptable 08048ADF case 6

loc_8048B03:                            ; CODE XREF: phase3+49↑j
                                        ; DATA XREF: .rodata:off_8049AE8↓o
                sub     [ebp+var_C], 68h ; jumptable 08048ADF case 7
                jmp     short loc_8048B0E
; ---------------------------------------------------------------------------

loc_8048B09:                            ; CODE XREF: phase3+40↑j
                call    explode_bomb    ; jumptable 08048ADF default case
; ---------------------------------------------------------------------------

loc_8048B0E:                            ; CODE XREF: phase3+71↑j
                mov     eax, [ebp+var_14]
                cmp     eax, 5
                jg      short loc_8048B1E

                mov     eax, [ebp+var_18]
                cmp     [ebp+var_C], eax
                jz      short loc_8048B23

loc_8048B1E:                            ; CODE XREF: phase3+7E↑j
                call    explode_bomb
; ---------------------------------------------------------------------------

loc_8048B23:                            ; CODE XREF: phase3+86↑j
                nop
                leave
                retn
phase3          endp
```

- **Mã giả:**

```
int __cdecl phase3(int a1)
{
  int result; // eax@14
  int v2; // [sp+0h] [bp-18h]@1
  int v3; // [sp+4h] [bp-14h]@1
  int v4; // [sp+8h] [bp-10h]@1
  int v5; // [sp+Ch] [bp-Ch]@1

  v5 = 0;
  v4 = 0;
  v4 = __isoc99_sscanf(a1, "%d %d", &v3, &v2);
  if ( v4 <= 1 )
    explode_bomb();
  switch ( v3 )
  {
    case 0:
      v5 -= 104;
      goto LABEL_5;
    case 1:
LABEL_5:
      v5 -= 611;
      goto LABEL_6;
    case 2:
LABEL_6:
      v5 += 56;
      goto LABEL_7;
    case 3:
LABEL_7:
      v5 -= 905;
      goto LABEL_8;
    case 4:
LABEL_8:
      v5 += 104;
      goto LABEL_9;
    case 5:
LABEL_9:
      v5 -= 104;
      goto LABEL_10;
    case 6:
LABEL_10:
      v5 += 104;
      break;
    case 7:
      break;
    default:
      explode_bomb();
      return result;
  }
  v5 -= 104;
  if ( v3 > 5 || (result = v2, v5 != v2) )
    explode_bomb();
  return result;
```

- **Định dạng đầu vào:** là 2 số nguyên (v3 và v2)
- **Điều kiện ràng buộc:**
  + Biến v4 lưu số lượng số đọc được từ input. Câu lệnh if so sánh v4 <= 1, Để hàm không nổ bom thì v4 phải lớn hơn 1 tức input cần nhập vào phải đúng 2 số nguyên.
  + Số nguyên đầu tiên nhập vào (v3) phải nằm trong đoạn [0; 5] (v3 <= 5).
  + Số nguyên thứ 2 (v2) phải bằng giá trị cuối cùng của v5 sau quá trình tính toán (v2 = v5).
- **Kết luận về đầu vào:**
  + Input cần nhập vào là 2 số nguyên (v3 và v2).
  + Số thứ nhất (v3) phải là một số nguyên bất kỳ từ 0 đến 5
  + Số thứ 2 (v2) phải bằng v5 (v2 = v5).
- **Cách tìm được đáp án nộp trên vlab:**
  + Nhìn vào mã giả pha 3, ta nhìn thấy đầu vào input là a1 có kiểu dữ liệu là int với dạng chuỗi gồm 2 kí tự và đưa số này vào v3 và v2.
  + Ở đầu đoạn code, v5 = 0, v4 = 0 được khai báo và câu lệnh tiếp theo đếm xem đã đọc thành công bao nhiêu số từ input và gán vào v4. Câu if tiếp theo kiểm tra v4 <= 1, nếu thỏa mãn thì gọi hàm explode_bomb(). ⇒ Câu if phải sai, tức là lệnh __sscanf đọc được ít nhất 2 số từ input.
  + Quan sát câu if ở cuối, ta thấy nếu v3 > 5 or với kết quả của điều kiện v5 != v2. Nếu thỏa if trên thì sẽ gọi hàm explode_bomb() ⇒ Câu if trên phải sai ở cả 2 vế nên ta có thể suy ra ràng buộc v3 <= 5 và v5 = v2.
  + Tiếp đến là switch case để so sánh giá trị của v3. Ta đã có ràng buộc v3 <= 5 nên theo logic của cấu trúc switch case ở trên thì với mỗi giá trị v3 nằm trong đoạn [0; 5] sẽ có giá trị của v5 tương ứng và v5 phải bằng v2, tức kí tự thứ 2 của input.

    ⇒ Ta suy ra được v5 như sau:

    + v3 = 0 → v5 = -1564
    + v3 = 1 → v5 = -1460
    + v3 = 2 → v5 = -849
    + v3 = 3 → v5 = -905
    + v3 = 4 → v5 = 0
    + v3 = 5 → v5 = -104

Vậy các input thỏa mãn là: (0 -1564), (1 -1460), (2 -849), (3 -905), (4 0), (5 -104).

  + Do yêu cầu của vLab nên hai số cần tìm là: **0 -1564**
- **Kết quả chạy:**
  + Với input là 0 -1564:

```
[*] Phase 3
- Hint: Many cases make everything so confusing.
0 -1564
You've beaten another phase, that's great. What about the fourth one?
```

  + Với các input còn lại:

```
[*] Phase 3
- Hint: Many cases make everything so confusing.
4 0
You've beaten another phase, that's great. What about the fourth one?
```

```
[*] Phase 3
- Hint: Many cases make everything so confusing.
1 -1460
You've beaten another phase, that's great. What about the fourth one?
```

```
[*] Phase 3
- Hint: Many cases make everything so confusing.
2 -849
You've beaten another phase, that's great. What about the fourth one?
```

```
[*] Phase 3
- Hint: Many cases make everything so confusing.
3 -905
You've beaten another phase, that's great. What about the fourth one?
```

```
[*] Phase 3
- Hint: Many cases make everything so confusing.
5 -104
You've beaten another phase, that's great. What about the fourth one?
```

4. Pha 4

- **Mã Assembly:**

```
; =============== S U B R O U T I N E ===========================

; Attributes: bp-based frame

                public phase4
phase4          proc near               ; CODE XREF: main+149↓p

var_1C          = dword ptr -1Ch
var_18          = dword ptr -18h
var_14          = dword ptr -14h
var_10          = dword ptr -10h
var_C           = dword ptr -0Ch
arg_0           = dword ptr  8

                push    ebp
                mov     ebp, esp
                sub     esp, 28h
                lea     eax, [ebp+var_1C]
                push    eax
                lea     eax, [ebp+var_18]
                push    eax
                push    offset aDD      ; "%d %d"
                push    [ebp+arg_0]
                call    ___isoc99_sscanf
                add     esp, 10h
                mov     [ebp+var_C], eax
```

```
                cmp     [ebp+var_C], 2
                jnz     short loc_8048BCD
                mov     eax, [ebp+var_18]
                test    eax, eax
                js      short loc_8048BCD
                mov     eax, [ebp+var_18]
                cmp     eax, 0Eh
                jle     short loc_8048BD2

loc_8048BCD:                            ; CODE XREF: phase4+25↑j
                                        ; phase4+2C↑j
                call    explode_bomb
; ---------------------------------------------------------------------------

loc_8048BD2:                            ; CODE XREF: phase4+34↑j
                mov     [ebp+var_10], 4
                mov     eax, [ebp+var_18]
                sub     esp, 4
                push    0Eh
                push    0
                push    eax
                call    func4
                add     esp, 10h
                mov     [ebp+var_14], eax
                mov     eax, [ebp+var_14]
                cmp     eax, [ebp+var_10]

                jz      short loc_8048C04

loc_8048BFF:                            ; CODE XREF: phase4+5E↑j
                call    explode_bomb
; ---------------------------------------------------------------------------

loc_8048C04:                            ; CODE XREF: phase4+66↑j
                nop
                leave
                retn
phase4          endp
```

- **Mã giả:**

```
1  int __cdecl phase4(int a1)
2  {
3    int result; // eax@6
4    int v2; // [sp+Ch] [bp-1Ch]@1
5    int v3; // [sp+10h] [bp-18h]@1
6    int v4; // [sp+14h] [bp-14h]@5
7    int v5; // [sp+18h] [bp-10h]@5
8    int v6; // [sp+1Ch] [bp-Ch]@1
9
10   v6 = __isoc99_sscanf(a1, "%d %d", &v3, &v2);
11   if ( v6 != 2 || v3 < 0 || v3 > 14 )
12     explode_bomb();
13   v5 = 4;
14   v4 = func4(v3, 0, 14);
15   if ( v4 != v5 || (result = v2, v2 != v5) )
16     explode_bomb();
17   return result;
18 }
```

- **Định dạng đầu vào:** là 2 số nguyên (v3 và v2)
- **Điều kiện ràng buộc:**
  - + Biến v6 lưu số lượng số đọc được từ input. Câu lệnh if kiểm tra v6 != 2, Để hàm không nổ bom thì v6 phải bằng 2.
  - + Số nguyên đầu tiên nhập vào (v3) phải lớn hơn 0 và nhỏ hơn 14 (0 < v3 < 14)
  - + Số nguyên thứ 2 (v2) phải bằng 4 (v2 = 4)

- **Kết luận về đầu vào:**
  - + Input cần nhập vào là 2 số nguyên (v3 và v2).
  - + Số thứ nhất (v3) phải là một số nguyên nằm trong khoảng (0, 14) sao cho khi đưa vào hàm func4() cùng với tham số 0 và 14 thì kết quả trả về phải bằng 4 để thỏa mãn v4 = 4.
  - + Số thứ 2 (v2) phải bằng 4.
- **Cách tìm được đáp án nộp trên vlab:**
  - + Nhìn vào mã giả pha 4, ta nhìn thấy đầu vào input là a1 có kiểu dữ liệu là int với dạng chuỗi gồm 2 kí tự và đưa số này vào v3 và v2.
  - + Câu lệnh tiếp theo đếm xem đã đọc thành công bao nhiêu số từ input và gán vào v6. Câu if tiếp theo kiểm tra điều kiện v6 != 2 or v3 < 0 or v3 > 14. Nếu thỏa mãn thì chương trình sẽ gọi hàm explode_bomb(). ⇒ Câu if trên phải sai ở cả 3 vế nên ta có thể suy ra v6 = 2 có nghĩa là đọc được đúng 2 số từ input và v3 phải nằm trong khoảng (0, 14).
  - + Quan sát câu if ở cuối, ta thấy nếu v4 != v5 or với kết quả của điều kiện v5 != v2. Nếu thỏa if trên thì sẽ gọi hàm explode_bomb() ⇒ Câu if trên phải sai ở cả 2 vế. Nên ta suy ra, v4 = v5 và v2 = v5 mà v5 đã được khai báo ở trên có giá trị bằng 4 ⇒ v4 = 4 và v2 = 4. Ta đã có được ký tự thứ 2, v2 = 4.
  - + Với v4 = 4 ta có thể tìm ngược v3 thông qua câu lệnh v4 = func4(v3, 0, 14). Mô phỏng lại hoạt động của hàm func4() bằng C để tìm v3 thỏa mãn v4 = 4 với biến i là v3:

```c
#include <iostream>
using namespace std;
int func4(int a1, int a2, int a3)
{
  int result; // eax@2
  int v4; // [sp+Ch] [bp-Ch]@1

  v4 = (a3 - a2) / 2 + a2;
  if ( v4 <= a1 )
  {
    if ( v4 >= a1 )
      result = 0;
    else
      result = 2 * func4(a1, v4 + 1, a3) + 1;
  }
  else
  {
    result = 2 * func4(a1, a2, v4 - 1);
  }
  return result;
}
```

```c
int main()
{
    for(int i = 0; i <= 14; i++)
    {
        int v4 = func4(i,0,14);
        if (v4 == 4)
        cout << i << " " << v4;
    }
    return 0;
}
```

```
C:\Users\Admin\Desktop\_STI
2 4
Process returned 0 (0x0)   execution time : 0.021 s
Press any key to continue.
```

→ Ta thu được kết quả v3 = 2. Ghép v3 vừa tìm được với v2 ta sẽ có được chuỗi đáp án.

Vậy input cần tìm là: **2 4**

- **Kết quả chạy:**

```
[*] Phase 4
- Hint: Let's dig in to recursive function :)
2 4
Congrats! You've found another correct input. Moving ahead!
```

5. Pha 5

- **Mã assembly:**

```
; int __cdecl phase5(int)
                public phase5
phase5          proc near               ; CODE XREF: main+174+p

var_20          = dword ptr -20h
var_1C          = dword ptr -1Ch
var_18          = dword ptr -18h
var_14          = dword ptr -14h
var_10          = dword ptr -10h
var_C           = dword ptr -0Ch
arg_0           = dword ptr  8

; __unwind {
                push    ebp
                mov     ebp, esp
                sub     esp, 28h
                lea     eax, [ebp+var_20]
                push    eax
                lea     eax, [ebp+var_1C]
                push    eax
                push    offset aDD      ; "%d %d"
                push    [ebp+arg_0]
                call    ___isoc99_sscanf
                add     esp, 10h
                mov     [ebp+var_14], eax
                cmp     [ebp+var_14], 1
                jg      short loc_8048C33
                call    explode_bomb
; ---------------------------------------------------------------------------

loc_8048C33:                            ; CODE XREF: phase5+25+j
                mov     eax, [ebp+var_1C]
                and     eax, 0Fh
                mov     [ebp+var_1C], eax
                mov     eax, [ebp+var_1C]
                mov     [ebp+var_18], eax
```

```
                mov     [ebp+var_C], 0
                mov     [ebp+var_10], 0
                jmp     short loc_8048C69
; ---------------------------------------------------------------------------

loc_8048C52:                            ; CODE XREF: phase5+68↑j
                add     [ebp+var_C], 1
                mov     eax, [ebp+var_1C]
                mov     eax, array_3854[eax*4]
                mov     [ebp+var_1C], eax
                mov     eax, [ebp+var_1C]
                add     [ebp+var_10], eax

loc_8048C69:                            ; CODE XREF: phase5+49↑j
                mov     eax, [ebp+var_1C]
                cmp     eax, 0Fh
                jnz     short loc_8048C52
                cmp     [ebp+var_C], 0Ah
                jnz     short loc_8048C7F
                mov     eax, [ebp+var_20]
                cmp     [ebp+var_10], eax
                jz      short loc_8048C84

loc_8048C7F:                            ; CODE XREF: phase5+6E↑j
                call    explode_bomb
; ---------------------------------------------------------------------------

loc_8048C84:                            ; CODE XREF: phase5+76↑j
                nop
                leave
                retn
; } // starts at 8048C07
phase5          endp
```

- **Mã giả:**

```
int __cdecl phase5(int a1)
{
  int result; // eax
  int v2; // [esp+8h] [ebp-20h] BYREF
  _DWORD v3[2]; // [esp+Ch] [ebp-1Ch] BYREF
  int v4; // [esp+14h] [ebp-14h]
  int v5; // [esp+18h] [ebp-10h]
  int v6; // [esp+1Ch] [ebp-Ch]

  v4 = __isoc99_sscanf(a1, "%d %d", v3, &v2);
  if ( v4 <= 1 )
    explode_bomb();
  v3[0] = v3[0] & 0xF;
  v3[1] = v3[0];
  v6 = 0;
  v5 = 0;
  while ( v3[0] != 15 )
  {
    ++v6;
    v3[0] = array_3854[v3[0]];
    v5 += v3[0];
  }
  if ( v6 != 10 || (result = v2, v5 != v2) )
    explode_bomb();
  return result;
}
```

- **Định dạng nhập vào:** 2 số nguyên (v3[0] và v2).
- **Điều kiện ràng buộc giữa các phần tử nhập vào:** không có.
- **Kết luận về đầu vào:** sau khi nhập, hàm chỉ lấy 4 bit thấp nhất của v3[0] (bằng phép AND: v3[0] = v3[0] & 0xF), nên **mọi** số có dạng 16n + x đều có thể là v3[0]. Ta chỉ cần tìm x và v2 để có một cặp số nhập vào thỏa mãn rồi suy ra các cặp khác.
- **Cách tìm đáp án nộp trên vLab:**
  + Ta thấy, sau khi lấy 4 bit cuối của v3[0], hàm bắt đầu vòng lặp: nếu x khác 15 thì tăng biến đếm v6, gán lại x = array_3854[x] rồi cộng x vừa gán vào v5:

```
v6 = 0;
v5 = 0;
while ( v3[0] != 15 )
{
  ++v6;
  v3[0] = array_3854[v3[0]];
  v5 += v3[0];
}
```

Trong đó, array_3854 là mảng số nguyên gồm 16 phần tử:

```
; int array_3854[16]
array_3854      dd 0Ah                  ; DATA XREF: phase5+52↑r
                db 2,0,0,0
                db 0Eh,0,0,0
                db 7,0,0,0
                db 8,0,0,0
                db 0Ch,0,0,0
                db 0Fh,0,0,0
                db 0Bh,0,0,0
                db 0,0,0,0
                db 4,0,0,0
                db 1,0,0,0
                db 0Dh,0,0,0
                db 3,0,0,0
                db 9,0,0,0
                db 6,0,0,0
                db 5,0,0,0
_data           ends
```

+ Nếu v5 khác v2 hoặc v6 khác 10 thì cho nổ bom:

```
if ( v6 != 10 || (result = v2, v5 != v2) )
  explode_bomb();
return result;
```

Do đó ta cần tìm x sao cho vòng lặp thực hiện đúng 10 lần và v2 = v5. Ta mô phỏng lại hoạt động của hàm bằng C để tìm x và v2:

```c
#include <stdio.h>

int
main(void)
{
        int array[] = { 10, 2, 14, 7, 8, 12, 15, 11, 0, 4, 1, 13, 3, 9, 6, 5 };
        for (int i = 0; i <= 15; i++) {
                int v3 = i;
                int v6 = 0;
                int v5 = 0;
                while (v3 != 15) {
                        ++v6;
                        v3 = array[v3];
                        v5 += v3;
                }
                if (v6 == 10)
                        printf("%d %d\n", i, v5);
        }
        return 0;
}
```

```
tch:~/build/tmp/sp % ./phase5
13 69
tch:~/build/tmp/sp %
```

+ Do yêu cầu của vLab nên hai số cần tìm là: **13 69**

- **Kết quả chạy:**
  + Với cặp 13, 69:

```
[*] Phase 5
-Hint: No hint is also a hint :)
13 69
Awesome! Only one phase left!
```

+ Với cặp -371, 69:

```
[*] Phase 5
-Hint: No hint is also a hint :)
-371 69
Awesome! Only one phase left!
```

+ Với cặp 749, 69:

```
[*] Phase 5
-Hint: No hint is also a hint :)
749 69
Awesome! Only one phase left!
```

6. Pha 6

- Mã assembly:

```
                public phase6
phase6          proc near               ; CODE XREF: main+19F↓p

var_4C          = dword ptr -4Ch
var_34          = dword ptr -34h
var_1C          = dword ptr -1Ch
var_18          = dword ptr -18h
var_14          = dword ptr -14h
var_10          = dword ptr -10h
var_C           = dword ptr -0Ch
arg_0           = dword ptr  8

; __unwind {
                push    ebp
                mov     ebp, esp
                sub     esp, 58h
                mov     [ebp+var_18], offset node1
                lea     eax, [ebp+var_34]
                add     eax, 14h
                push    eax
                lea     eax, [ebp+var_34]
                add     eax, 10h
                push    eax
                lea     eax, [ebp+var_34]
                add     eax, 0Ch
                push    eax
                lea     eax, [ebp+var_34]
                add     eax, 8
                push    eax
                lea     eax, [ebp+var_34]
                add     eax, 4
                push    eax
                lea     eax, [ebp+var_34]
                push    eax
                push    offset aDDDDDD  ; "%d %d %d %d %d %d"
```

```
                push    [ebp+arg_0]
                call    ___isoc99_sscanf
                add     esp, 20h
                mov     [ebp+var_1C], eax
                cmp     [ebp+var_1C], 6
                jz      short loc_8048CD9
                call    explode_bomb
; ---------------------------------------------------------------------------

loc_8048CD9:                            ; CODE XREF: phase6+4B↑j
                mov     [ebp+var_10], 0
                jmp     short loc_8048D2E
; ---------------------------------------------------------------------------

loc_8048CE2:                            ; CODE XREF: phase6+AB↑j
                mov     eax, [ebp+var_10]
                mov     eax, [ebp+eax*4+var_34]
                test    eax, eax
                jle     short loc_8048CF9
                mov     eax, [ebp+var_10]
                mov     eax, [ebp+eax*4+var_34]
                cmp     eax, 6
                jle     short loc_8048CFE

loc_8048CF9:                            ; CODE XREF: phase6+64↑j
                call    explode_bomb
; ---------------------------------------------------------------------------

loc_8048CFE:                            ; CODE XREF: phase6+70↑j
                mov     eax, [ebp+var_10]
                add     eax, 1
                mov     [ebp+var_14], eax
                jmp     short loc_8048D24
; ---------------------------------------------------------------------------


loc_8048D09:                            ; CODE XREF: phase6+A1↑j
                mov     eax, [ebp+var_10]
                mov     edx, [ebp+eax*4+var_34]
                mov     eax, [ebp+var_14]
                mov     eax, [ebp+eax*4+var_34]
                cmp     edx, eax
                jnz     short loc_8048D20
                call    explode_bomb
; ---------------------------------------------------------------------------

loc_8048D20:                            ; CODE XREF: phase6+92↑j
                add     [ebp+var_14], 1

loc_8048D24:                            ; CODE XREF: phase6+80↑j
                cmp     [ebp+var_14], 5
                jle     short loc_8048D09
                add     [ebp+var_10], 1

loc_8048D2E:                            ; CODE XREF: phase6+59↑j
                cmp     [ebp+var_10], 5
                jle     short loc_8048CE2
                mov     [ebp+var_10], 0
                jmp     short loc_8048D56
; ---------------------------------------------------------------------------

loc_8048D3D:                            ; CODE XREF: phase6+D3↑j
                mov     eax, [ebp+var_10]
                mov     eax, [ebp+eax*4+var_34]
                mov     edx, 7
                sub     edx, eax
                mov     eax, [ebp+var_10]
                mov     [ebp+eax*4+var_34], edx
                add     [ebp+var_10], 1
```

```
loc_8048D56:                            ; CODE XREF: phase6+B4↑j
                cmp     [ebp+var_10], 5
                jle     short loc_8048D3D
                mov     [ebp+var_10], 0
                jmp     short loc_8048D9B
; ---------------------------------------------------------------------------

loc_8048D65:                            ; CODE XREF: phase6+118↑j
                mov     eax, [ebp+var_18]
                mov     [ebp+var_C], eax
                mov     [ebp+var_14], 1
                jmp     short loc_8048D81
; ---------------------------------------------------------------------------

loc_8048D74:                            ; CODE XREF: phase6+104↑j
                mov     eax, [ebp+var_C]
                mov     eax, [eax+8]
                mov     [ebp+var_C], eax
                add     [ebp+var_14], 1

loc_8048D81:                            ; CODE XREF: phase6+EB↑j
                mov     eax, [ebp+var_10]
                mov     eax, [ebp+eax*4+var_34]
                cmp     eax, [ebp+var_14]
                jg      short loc_8048D74
                mov     eax, [ebp+var_10]
                mov     edx, [ebp+var_C]
                mov     [ebp+eax*4+var_4C], edx
                add     [ebp+var_10], 1

loc_8048D9B:                            ; CODE XREF: phase6+DC↑j
                cmp     [ebp+var_10], 5
                jle     short loc_8048D65
                mov     eax, [ebp+var_4C]
                mov     [ebp+var_18], eax

                mov     eax, [ebp+var_18]
                mov     [ebp+var_C], eax
                mov     [ebp+var_10], 1
                jmp     short loc_8048DD0
; ---------------------------------------------------------------------------

loc_8048DB6:                            ; CODE XREF: phase6+14D↑j
                mov     eax, [ebp+var_10]
                mov     edx, [ebp+eax*4+var_4C]
                mov     eax, [ebp+var_C]
                mov     [eax+8], edx
                mov     eax, [ebp+var_C]
                mov     eax, [eax+8]
                mov     [ebp+var_C], eax
                add     [ebp+var_10], 1

loc_8048DD0:                            ; CODE XREF: phase6+12D↑j
                cmp     [ebp+var_10], 5
                jle     short loc_8048DB6
                mov     eax, [ebp+var_C]
                mov     dword ptr [eax+8], 0
                mov     eax, [ebp+var_18]
                mov     [ebp+var_C], eax
                mov     [ebp+var_10], 0
                jmp     short loc_8048E12
; ---------------------------------------------------------------------------

loc_8048DEF:                            ; CODE XREF: phase6+18F↑j
                mov     eax, [ebp+var_C]
                mov     edx, [eax]
                mov     eax, [ebp+var_C]
                mov     eax, [eax+8]
                mov     eax, [eax]
                cmp     edx, eax
                jge     short loc_8048E05
```

```
                call    explode_bomb
; ---------------------------------------------------------------------------

loc_8048E05:                            ; CODE XREF: phase6+177↑j
                mov     eax, [ebp+var_C]
                mov     eax, [eax+8]
                mov     [ebp+var_C], eax
                add     [ebp+var_10], 1

loc_8048E12:                            ; CODE XREF: phase6+166↑j
                cmp     [ebp+var_10], 4
                jle     short loc_8048DEF
                nop
                leave
                retn
; } // starts at 8048C87
phase6          endp
```

- **Mã giả:**

```
_DWORD *__cdecl phase6(int a1)
{
  _DWORD *result; // eax
  _DWORD v2[6]; // [esp+Ch] [ebp-4Ch]
  int v3; // [esp+24h] [ebp-34h] BYREF
  char v4; // [esp+28h] [ebp-30h] BYREF
  char v5; // [esp+2Ch] [ebp-2Ch] BYREF
  char v6; // [esp+30h] [ebp-28h] BYREF
  char v7; // [esp+34h] [ebp-24h] BYREF
  char v8; // [esp+38h] [ebp-20h] BYREF
  int v9; // [esp+3Ch] [ebp-1Ch]
  _DWORD *v10; // [esp+40h] [ebp-18h]
  int j; // [esp+44h] [ebp-14h]
  int i; // [esp+48h] [ebp-10h]
  _DWORD *v13; // [esp+4Ch] [ebp-Ch]

  v10 = &node1;
  v9 = __isoc99_sscanf(a1, "%d %d %d %d %d %d", &v3, &v4, &v5, &v6, &v7, &v8);
  if ( v9 != 6 )
    explode_bomb();
  for ( i = 0; i <= 5; ++i )
  {
    if ( *(&v3 + i) <= 0 || *(&v3 + i) > 6 )
      explode_bomb();
    for ( j = i + 1; j <= 5; ++j )
    {
      if ( *(&v3 + i) == *(&v3 + j) )
        explode_bomb();
    }
  }

  for ( i = 0; i <= 5; ++i )
    *(&v3 + i) = 7 - *(&v3 + i);
  for ( i = 0; i <= 5; ++i )
  {
    v13 = v10;
    for ( j = 1; *(&v3 + i) > j; ++j )
      v13 = (_DWORD *)v13[2];
    v2[i] = v13;
  }
  v10 = (_DWORD *)v2[0];
  v13 = (_DWORD *)v2[0];
  for ( i = 1; i <= 5; ++i )
  {
    v13[2] = v2[i];
    v13 = (_DWORD *)v13[2];
  }
  v13[2] = 0;
  result = v10;
  v13 = v10;
  for ( i = 0; i <= 4; ++i )
  {
    if ( *v13 < *(_DWORD *)v13[2] )
      explode_bomb();
    result = (_DWORD *)v13[2];
    v13 = result;
  }
  return result;
}
```

- **Định dạng nhập vào:** 6 số nguyên (v3, v4, v5, v6, v7, v8)
- **Điều kiện ràng buộc:** 6 số thuộc đoạn từ 1 tới 6, không trùng nhau.
  - + Kích nổ bom nếu 1 trong 2 điều kiện trên không thỏa mãn:

```
v10 = &node1;
v9 = __isoc99_sscanf(a1, "%d %d %d %d %d %d", &v3, &v4, &v5, &v6, &v7, &v8);
if ( v9 != 6 )
  explode_bomb();
for ( i = 0; i <= 5; ++i )
{
  if ( *(&v3 + i) <= 0 || *(&v3 + i) > 6 )
    explode_bomb();
  for ( j = i + 1; j <= 5; ++j )
  {
    if ( *(&v3 + i) == *(&v3 + j) )
      explode_bomb();
  }
}
```

- **Kết luận về đầu vào:** do điều kiện ràng buộc ở trên, có thể có tối đa 6! = 720 tổ hợp 6 số thỏa mãn đầu vào.
- **Cách tìm đáp án nộp trên vLab:**
  + Đầu chương trình, ta thấy v10 được gán tới địa chỉ node1:

```
                public node6
node6           db '4',0,0,0,6,0,0,0,0,0,0,0
                public node5
node5           db 0BAh,2,0,0,5,0,0,0,0ECh,0C1h,4,8
                public node4
node4           db '~',1,0,0,4,0,0,0,0F8h,0C1h,4,8
                public node3
node3           db 'p',1,0,0,3,0,0,0,4,0C2h,4,8
                public node2
node2           db 'p',2,0,0,2,0,0,0,10h,0C2h,4,8
                public node1
node1           db 95h,3,0,0,1,0,0,0,1Ch,0C2h,4,8
                                        ; DATA XREF: phase6+6+o
```

Ta thấy trên node1 còn có 5 node khác từ node2 tới node6. Mỗi node đều có chiều dài 12 byte. Ta thấy bên dưới mã giả chỉ truy xuất các phần tử [0], [2]:

```
  for ( j = 1; *(&v3 + i) >
    v13 = (_DWORD *)v13[2];
  v2[i] = v13;
}
v10 = (_DWORD *)v2[0];
v13 = (_DWORD *)v2[0];
for ( i = 1; i <= 5; ++i )
{
  v13[2] = v2[i];
  v13 = (_DWORD *)v13[2];
}
v13[2] = 0;
```

Do đó chúng ta có thể dự đoán mỗi node này gồm 3 phần tử:

(1) 4 byte đầu chứa 1 giá trị có thể là dữ liệu cần lưu.

(2) 4 byte giữa có giá trị trùng với số trong tên node, có thể là số thứ tự

(3) 4 byte cuối, ta thấy 4 byte cuối của node1 là 0x0804C21C trùng với địa chỉ node2, do đó rất có thể là địa chỉ node tiếp theo.

  + Lần lượt gọi các phần tử trên là Data, Index và PNext, ta được bảng sau:

	Địa chỉ	Data	Index	PNext
node1	0x0804C228	0x395	1	0x0804C21C
node2	0x0804C21C	0x270	2	0x0804C210
node3	0x0804C210	0x170	3	0x0804C204
node4	0x0804C204	0x17E	4	0x0804C1F8
node5	0x0804C1F8	0x2BA	5	0x0804C1EC

node6	0x0804C1EC	0x34	6	0

+ Do các số được lưu liền kề nhau, từ đây ta có thể coi như dữ liệu nhập vào được lưu vào 1 mảng có địa chỉ bắt đầu là địa chỉ của v3.

```
for ( i = 0; i <= 5; ++i )
  *(&v3 + i) = 7 - *(&v3 + i);
for ( i = 0; i <= 5; ++i )
{
  v13 = v10;
  for ( j = 1; *(&v3 + i) > j; ++j )
    v13 = (_DWORD *)v13[2];
  v2[i] = v13;
}
v10 = (_DWORD *)v2[0];
v13 = (_DWORD *)v2[0];
for ( i = 1; i <= 5; ++i )
{
  v13[2] = v2[i];
  v13 = (_DWORD *)v13[2];
}
v13[2] = 0;
result = v10;
v13 = v10;
for ( i = 0; i <= 4; ++i )
{
  if ( *v13 < *(_DWORD *)v13[2] )
    explode_bomb();
  result = (_DWORD *)v13[2];
  v13 = result;
}
return result;
}
```

+ Chương trình gán giá trị mới cho mỗi số mới nhập vào bằng 7 - <giá trị cũ>, sắp xếp lại các node theo thứ tự vừa có, rồi duyệt danh sách, kiểm tra nếu Data trong node hiện tại bé hơn node sau thì cho nổ bom.

  Do đó, ta cần chọn thứ tự sao cho Data của các node được sắp từ lớn tới bé. Với thứ tự 1-5-2-4-3-6 ta có 0x395 > 0x2BA > 0x270 > 0x17E > 0x170 > 0x34 như mong muốn, lấy 7 trừ lại ta được thứ tự cần tìm là **6 2 5 3 4 1**.

- **Kết quả thực thi:**

```
[*] Phase 6
-Hint: Can you master single linked list?
6 2 5 3 4 1
Amazing bomb solvers, the bomb has been deactived. Enjoy your day :))
tch:~/build/tmp/sp %
```