Rendu TP1

1) Phase 1

On va tout d'abord regarder en détailler le code assembleur de la phase_1:

```
(gdb) disas phase_1
Dump of assembler code for function phase_1:
   0x0000000000002439 <+0>:     sub    $0x8,%rsp
   0x000000000000243d <+4>:     lea    0x1d0c(%rip),%rsi        # 0x4150
   0x0000000000002444 <+11>:    call   0x290e <strings_not_equal>
   0x0000000000002449 <+16>:    test   %eax,%eax
   0x000000000000244b <+18>:    jne    0x2452 <phase_1+25>
   0x000000000000244d <+20>:    add    $0x8,%rsp
   0x0000000000002451 <+24>:    ret
   0x0000000000002452 <+25>:    call   0x2b53 <explode_bomb>
   0x0000000000002457 <+30>:    jmp    0x244d <phase_1+20>
End of assembler dump.
```

Ainsi , on remarque que la phase_1 vérifiera juste si la chaine est égal à la chaine correspondante , sinon la bombe explosera , donc il suffira d'afficher la valeur de l'adresse 0x4150. On aura ainsi comme résultat : « For NASA, space is still a high priority. ».

```
(gdb) x/s 0x4150
0x4150: "For NASA, space is still a high priority."
```

Ainsi le résultat attendu de la phase_1 est: **For NASA, space is still a high priority.**

2) Phase 2

```
(gdb) disas phase_2
Dump of assembler code for function phase_2:
   0x0000000000002459 <+0>:     push   %rbp
   0x000000000000245a <+1>:     push   %rbx
   0x000000000000245b <+2>:     sub    $0x28,%rsp
   0x000000000000245f <+6>:     mov    %rsp,%rsi
   0x0000000000002462 <+9>:     call   0x2b8f <read_six_numbers>
   0x0000000000002467 <+14>:    cmpl   $0x1,(%rsp)
   0x000000000000246b <+18>:    jne    0x2477 <phase_2+30>
   0x000000000000246d <+20>:    mov    %rsp,%rbx
   0x0000000000002470 <+23>:    lea    0x14(%rsp),%rbp
   0x0000000000002475 <+28>:    jmp    0x2487 <phase_2+46>
   0x0000000000002477 <+30>:    call   0x2b53 <explode_bomb>
   0x000000000000247c <+35>:    jmp    0x246d <phase_2+20>
   0x000000000000247e <+37>:    add    $0x4,%rbx
   0x0000000000002482 <+41>:    cmp    %rbp,%rbx
   0x0000000000002485 <+44>:    je     0x2497 <phase_2+62>
   0x0000000000002487 <+46>:    mov    (%rbx),%eax
   0x0000000000002489 <+48>:    add    %eax,%eax
   0x000000000000248b <+50>:    cmp    %eax,0x4(%rbx)
   0x000000000000248e <+53>:    je     0x247e <phase_2+37>
   0x0000000000002490 <+55>:    call   0x2b53 <explode_bomb>
   0x0000000000002495 <+60>:    jmp    0x247e <phase_2+37>
   0x0000000000002497 <+62>:    add    $0x28,%rsp
   0x000000000000249b <+66>:    pop    %rbx
--Type <RET> for more, q to quit, c to continue without paging--c
   0x000000000000249c <+67>:    pop    %rbp
   0x000000000000249d <+68>:    ret
End of assembler dump.
```

On remarquera que la valeur de retour est sous forme de %d %d %d %d %d %d.
La fonction read_six_numbers lira six nombres à partir de l'entrée utilisateur et les stockera dans la pile. Après la lecture des nombres, on fera une première comparaison , où on comparera à l'adresse `0x2467 si` le premier élément du tableau est bien égal à `1`. Si le premier nombre n'est pas égal à `1`, la bombe explose .Si le premier nombre est égal à 1 , on parcourra les 5 nombres restant. Pour les 5 nombres restants , on parcourera une boucle , la boucle commence a l'adresse <+46>. A chaque itération de la boucle, on va additionner le nombre précédent , car à l'adresse <+48> , on fait « $eax,$eax ». Ensuite on vérifiera si la somme à bien été réussi à l'adresse <+50>, si c'est le cas on continue notre parcours sinon la bombe explose Puis on déplacera $rbp de 4 octets, jusqu'a ce qu'elle soit égal à $rbx. Avec cette logique on obtiendra les valeurs suivantes :

1. Valeur 1=1 (c'est forcément 1 vu la 1er comparaison faites)
2. Valeur 2=2 (Valeur1+Valeur1=2)
3. Valeur 3=4 (Valeur2+Valeur2=4)
4. Valeur 4=8 (Valeur3+Valeur3=8)
5. Valeur 5=16 (Valeur4+Valeur4=16)
6. Valeur 6=32 (Valeur5+Valeur6=32)

Ainsi le résultat attendu de la phase_2 est: **1 2 4 8 16 32**

3) Phase 3

```
(gdb) disas phase_3
Dump of assembler code for function phase_3:
   0x000000000000249e <+0>:     sub    $0x18,%rsp
   0x00000000000024a2 <+4>:     lea    0x7(%rsp),%rcx
   0x00000000000024a7 <+9>:     lea    0xc(%rsp),%rdx
   0x00000000000024ac <+14>:    lea    0x8(%rsp),%r8
   0x00000000000024b1 <+19>:    lea    0x1cee(%rip),%rsi        # 0x41a6
   0x00000000000024b8 <+26>:    mov    $0x0,%eax
   0x00000000000024bd <+31>:    call   0x2150 <__isoc99_sscanf@plt>
   0x00000000000024c2 <+36>:    cmp    $0x2,%eax
   0x00000000000024c5 <+39>:    jle    0x24e6 <phase_3+72>
   0x00000000000024c7 <+41>:    cmpl   $0x7,0xc(%rsp)
   0x00000000000024cc <+46>:    ja     0x25db <phase_3+317>
   0x00000000000024d2 <+52>:    mov    0xc(%rsp),%eax
   0x00000000000024d6 <+56>:    lea    0x1ce3(%rip),%rdx        # 0x41c0
   0x00000000000024dd <+63>:    movslq (%rdx,%rax,4),%rax
   0x00000000000024e1 <+67>:    add    %rdx,%rax
   0x00000000000024e4 <+70>:    jmp    *%rax
   0x00000000000024e6 <+72>:    call   0x2b53 <explode_bomb>
   0x00000000000024eb <+77>:    jmp    0x24c7 <phase_3+41>
   0x00000000000024ed <+79>:    mov    $0x71,%eax
   0x00000000000024f2 <+84>:    cmpl   $0x380,0x8(%rsp)
   0x00000000000024fa <+92>:    je     0x25e5 <phase_3+327>
   0x0000000000002500 <+98>:    call   0x2b53 <explode_bomb>
   0x0000000000002505 <+103>:   mov    $0x71,%eax
   0x000000000000250a <+108>:   jmp    0x25e5 <phase_3+327>
   0x000000000000250f <+113>:   mov    $0x69,%eax
   0x0000000000002514 <+118>:   cmpl   $0x47,0x8(%rsp)
   0x0000000000002519 <+123>:   je     0x25e5 <phase_3+327>
--Type <RET> for more, q to quit, c to continue without paging--c
   0x000000000000251f <+129>:   call   0x2b53 <explode_bomb>
   0x0000000000002524 <+134>:   mov    $0x69,%eax
   0x0000000000002529 <+139>:   jmp    0x25e5 <phase_3+327>
   0x000000000000252e <+144>:   mov    $0x69,%eax
   0x0000000000002533 <+149>:   cmpl   $0x114,0x8(%rsp)
   0x000000000000253b <+157>:   je     0x25e5 <phase_3+327>
```

```
0x000000000000253b <+157>:   je     0x25e5 <phase_3+327>
0x0000000000002541 <+163>:   call   0x2b53 <explode_bomb>
0x0000000000002546 <+168>:   mov    $0x69,%eax
0x000000000000254b <+173>:   jmp    0x25e5 <phase_3+327>
0x0000000000002550 <+178>:   mov    $0x65,%eax
0x0000000000002555 <+183>:   cmpl   $0x3e8,0x8(%rsp)
0x000000000000255d <+191>:   je     0x25e5 <phase_3+327>
0x0000000000002563 <+197>:   call   0x2b53 <explode_bomb>
0x0000000000002568 <+202>:   mov    $0x65,%eax
0x000000000000256d <+207>:   jmp    0x25e5 <phase_3+327>
0x000000000000256f <+209>:   mov    $0x67,%eax
0x0000000000002574 <+214>:   cmpl   $0x366,0x8(%rsp)
0x000000000000257c <+222>:   je     0x25e5 <phase_3+327>
0x000000000000257e <+224>:   call   0x2b53 <explode_bomb>
0x0000000000002583 <+229>:   mov    $0x67,%eax
0x0000000000002588 <+234>:   jmp    0x25e5 <phase_3+327>
0x000000000000258a <+236>:   mov    $0x6c,%eax
0x000000000000258f <+241>:   cmpl   $0x2dd,0x8(%rsp)
0x0000000000002597 <+249>:   je     0x25e5 <phase_3+327>
0x0000000000002599 <+251>:   call   0x2b53 <explode_bomb>
0x000000000000259e <+256>:   mov    $0x6c,%eax
0x00000000000025a3 <+261>:   jmp    0x25e5 <phase_3+327>
0x00000000000025a5 <+263>:   mov    $0x6f,%eax
0x00000000000025aa <+268>:   cmpl   $0xdd,0x8(%rsp)
0x00000000000025b2 <+276>:   je     0x25e5 <phase_3+327>
0x00000000000025b4 <+278>:   call   0x2b53 <explode_bomb>
0x00000000000025b9 <+283>:   mov    $0x6f,%eax
0x00000000000025be <+288>:   jmp    0x25e5 <phase_3+327>
0x00000000000025c0 <+290>:   mov    $0x6b,%eax
0x00000000000025c5 <+295>:   cmpl   $0x234,0x8(%rsp)
0x00000000000025cd <+303>:   je     0x25e5 <phase_3+327>
0x00000000000025cf <+305>:   call   0x2b53 <explode_bomb>
0x00000000000025d4 <+310>:   mov    $0x6b,%eax
0x00000000000025d9 <+315>:   jmp    0x25e5 <phase_3+327>
0x00000000000025db <+317>:   call   0x2b53 <explode_bomb>
0x00000000000025e0 <+322>:   mov    $0x77,%eax
0x00000000000025e5 <+327>:   cmp    %al,0x7(%rsp)
0x00000000000025e9 <+331>:   jne    0x25f0 <phase_3+338>
0x00000000000025eb <+333>:   add    $0x18,%rsp
0x00000000000025ef <+337>:   ret
0x00000000000025f0 <+338>:   call   0x2b53 <explode_bomb>
0x00000000000025f5 <+343>:   jmp    0x25eb <phase_3+333>
```

```
(gdb) x/s 0x41a6
0x41a6: "%d %c %d"
(gdb)
```

On sait que la valeur de retour sera un nombre suivi d’un char et un nombre.

On sait aussi que la valeur doit être inférieur ou égal à 7 , car à l'adresse <+47> , on fait une comparaison avec la 1er valeur et 7, donc si la 1er valeur est strictement supérieur à 7 la bombe explose , dans le cas contraire on peux continuer notre recherche.
Pour la suite du code , on remarquera qu'on a une série de comparaison , 8 au total , donc si la 8 ème comparison réussi la 1er valeur sera égal à 7 , la troisième valeur vaudra la valeur de la comparaison réussi, et la deuxième valeur sera égal au ASCII de la valeur.
En regardant la dernière comparaison <+327>, donc qui comparais $al à 0x65=105 en ASCII (comme j'avais désamorcer ma bombe la comparaison de fin à changer mais normalement j'avais à la place de 0x7 0x69), j'ai regardais les mov qui permettais d'avoir la valeur 105='i'.
Donc maintenant on sait que Valeur2='i', cherchons les Valeur1 et Valeur3.
On va regarder dans un premier temps les mov qui permettent d'avoir la valeur 0x69='i', puis on va regarder la comparaison qui suit , car on fait des mov dans le cas ou la comparaison ne réussit pas.
Donc la 3 ème comparaison compare la 3ème valeur avec 0x114=276 en décimal, et avant cette comparaison on a bien la Valeur2=0x69.
Ainsi avec cette logique on obtient les valeurs:

1. Valeur 1=2 (Le numéro de la comparaison qui nous a permis d'obtenir valeur2='i')
2. Valeur 2='i'
3. Valeur 3=276 (Résultat de la 3ème comparaison)

Ainsi le résultat attendu de la phase_3 est: **2 i 276.**

4) Phase 4

```
(gdb) disas phase_4
Dump of assembler code for function phase_4:
   0x000055555555662e <+0>:     sub    $0x18,%rsp
   0x0000555555556632 <+4>:     lea    0xc(%rsp),%rcx
   0x0000555555556637 <+9>:     lea    0x8(%rsp),%rdx
   0x000055555555663c <+14>:    lea    0x1e02(%rip),%rsi        # 0x555555558445
   0x0000555555556643 <+21>:    mov    $0x0,%eax
   0x0000555555556648 <+26>:    call   0x555555556150 <__isoc99_sscanf@plt>
   0x000055555555664d <+31>:    cmp    $0x2,%eax
   0x0000555555556650 <+34>:    jne    0x55555555665e <phase_4+48>
   0x0000555555556652 <+36>:    mov    0xc(%rsp),%eax
   0x0000555555556656 <+40>:    sub    $0x2,%eax
   0x0000555555556659 <+43>:    cmp    $0x2,%eax
   0x000055555555665c <+46>:    jbe    0x555555556663 <phase_4+53>
   0x000055555555665e <+48>:    call   0x555555556b53 <explode_bomb>
   0x0000555555556663 <+53>:    mov    0xc(%rsp),%esi
   0x0000555555556667 <+57>:    mov    $0x6,%edi
   0x000055555555666c <+62>:    call   0x5555555565f7 <func4>
   0x0000555555556671 <+67>:    cmp    %eax,0x8(%rsp)
   0x0000555555556675 <+71>:    jne    0x55555555667c <phase_4+78>
   0x0000555555556677 <+73>:    add    $0x18,%rsp
   0x000055555555667b <+77>:    ret
   0x000055555555667c <+78>:    call   0x555555556b53 <explode_bomb>
   0x0000555555556681 <+83>:    jmp    0x555555556677 <phase_4+73>
End of assembler dump.
```

```
(gdb) disas func4
Dump of assembler code for function func4:
   0x000055555555565f7 <+0>:     mov    $0x0,%eax
   0x000055555555565fc <+5>:     test   %edi,%edi
   0x000055555555565fe <+7>:     jle    0x55555555662d <func4+54>
   0x0000555555556600 <+9>:      push   %r12
   0x0000555555556602 <+11>:     push   %rbp
   0x0000555555556603 <+12>:     push   %rbx
   0x0000555555556604 <+13>:     mov    %edi,%ebx
   0x0000555555556606 <+15>:     mov    %esi,%ebp
   0x0000555555556608 <+17>:     mov    %esi,%eax
   0x000055555555560a <+19>:     cmp    $0x1,%edi
   0x000055555555560d <+22>:     je     0x555555556628 <func4+49>
   0x000055555555560f <+24>:     lea    -0x1(%rdi),%edi
   0x0000555555556612 <+27>:     call   0x55555555565f7 <func4>
   0x0000555555556617 <+32>:     lea    (%rax,%rbp,1),%r12d
   0x000055555555561b <+36>:     lea    -0x2(%rbx),%edi
   0x000055555555561e <+39>:     mov    %ebp,%esi
   0x0000555555556620 <+41>:     call   0x55555555565f7 <func4>
   0x0000555555556625 <+46>:     add    %r12d,%eax
   0x0000555555556628 <+49>:     pop    %rbx
   0x0000555555556629 <+50>:     pop    %rbp
   0x000055555555562a <+51>:     pop    %r12
   0x000055555555562c <+53>:     ret
   0x000055555555562d <+54>:     ret
End of assembler dump.
```

```
(gdb) x/s 0x555555558445
0x555555558445: "%d %d"
```

On sait que la valeur de retour sera deux nombre.
Ici c'est surtout la func4 qui va nous permettre de trouver le résultat final , car c'est là qu'on fera plusieurs appel récursif.Ainsi , on initialise $edi à 6 et $esi à 0, puis on appel la fonction `func4` est avec `edi` (premier argument) valant 6 et `esi` (deuxième argument) valant 0.
Lors du premier appel , func4 fera une seul opération jusqu'a ce que edi=0 :
1- edi-1 (car l'adresse<+19> ne passe pas donc %edi!=1, donc on continuera la récession jusqu'a ce edi=0) .
Ensuite elle revient en arrière et calcule `r12d = $rax + $rbp`, où $rax contient le résultat du précédent appel récursif et $rbp contient la valeur de $rsi (qui est 0 au premier appel).
Enfin, elle effectue un nouvel appel récursif avec $edi`-2`, et stocke ce résultat.
Ainsi en suivant ce fonctionnement, ces appels récursifs on obtiendra pour la phase_4 le résultat suivant : **40 2**

5) Phase 5

```
(gdb) disas phase_5
Dump of assembler code for function phase_5:
   0x0000555555556683 <+0>:     push   %rbx
   0x0000555555556684 <+1>:     mov    %rdi,%rbx
   0x0000555555556687 <+4>:     call   0x5555555568f1 <string_length>
   0x000055555555568c <+9>:     cmp    $0x6,%eax
   0x000055555555568f <+12>:    jne    0x5555555566bd <phase_5+58>
   0x0000555555556691 <+14>:    mov    %rbx,%rax
   0x0000555555556694 <+17>:    lea    0x6(%rbx),%rdi
   0x0000555555556698 <+21>:    mov    $0x0,%ecx
   0x000055555555569d <+26>:    lea    0x1b3c(%rip),%rsi        # 0x5555555581e0
 <array.0>
   0x00005555555566a4 <+33>:    movzbl (%rax),%edx
   0x00005555555566a7 <+36>:    and    $0xf,%edx
   0x00005555555566aa <+39>:    add    (%rsi,%rdx,4),%ecx
   0x00005555555566ad <+42>:    add    $0x1,%rax
   0x00005555555566b1 <+46>:    cmp    %rdi,%rax
   0x00005555555566b4 <+49>:    jne    0x5555555566a4 <phase_5+33>
   0x00005555555566b6 <+51>:    cmp    $0x31,%ecx
   0x00005555555566b9 <+54>:    jne    0x5555555566c4 <phase_5+65>
   0x00005555555566bb <+56>:    pop    %rbx
   0x00005555555566bc <+57>:    ret
   0x00005555555566bd <+58>:    call   0x555555556b53 <explode_bomb>
   0x00005555555566c2 <+63>:    jmp    0x555555556691 <phase_5+14>
   0x00005555555566c4 <+65>:    call   0x555555556b53 <explode_bomb>
   0x00005555555566c9 <+70>:    jmp    0x5555555566bb <phase_5+56>
End of assembler dump.
```

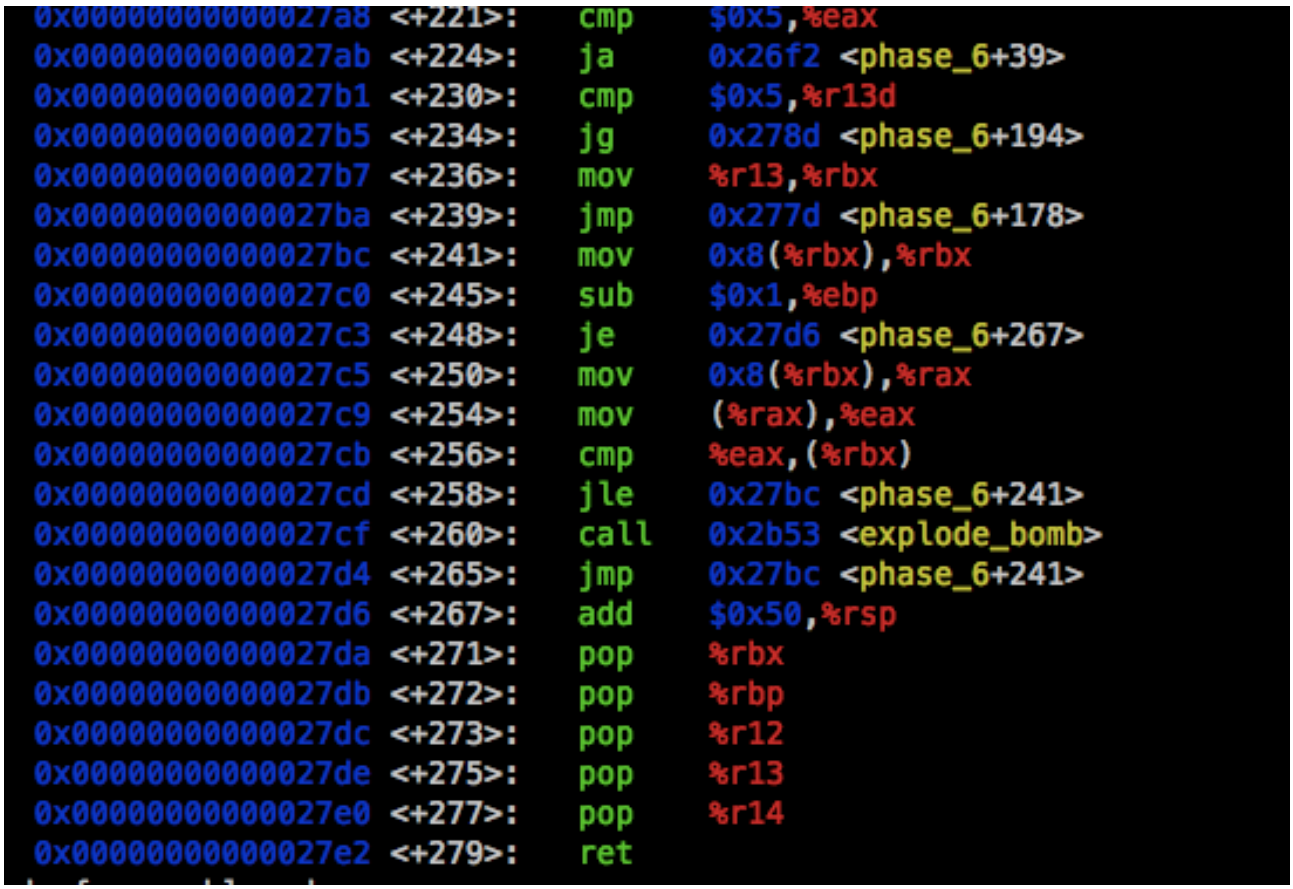
```
(gdb) x/15wd 0x5555555581e0
0x5555555581e0 <array.0>:       2       10      6       1
0x5555555581f0 <array.0+16>:    12      16      9       3
0x555555558200 <array.0+32>:    4       7       14      5
0x555555558210 <array.0+48>:    11      8       15
```

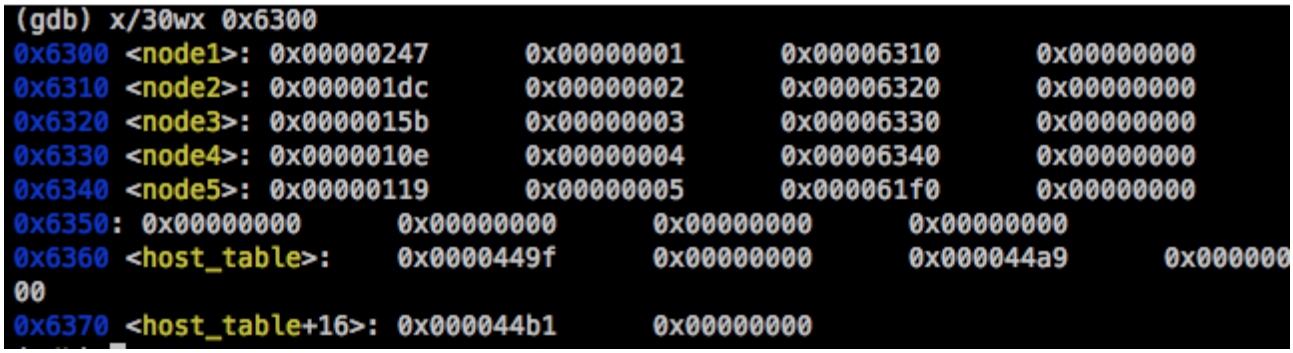
Le résultat attendue pour cette phase est une chaine de caractère de longueur 6 (comparaison à l'adresse <+9> si c'est pas égal cela bombe explose) . De plus , la somme des caractères doit être égal à 0x31=49 en décimal. Donc en regardant notre tableau , on devra trouver une combinaison qui soit égal à 49.

Ainsi le résultat attendu de la phase_5 est: **BCCEEF.**

6) Phase 6

```
(gdb) disas phase_6
Dump of assembler code for function phase_6:
   0x00000000000026cb <+0>:     push   %r14
   0x00000000000026cd <+2>:     push   %r13
   0x00000000000026cf <+4>:     push   %r12
   0x00000000000026d1 <+6>:     push   %rbp
   0x00000000000026d2 <+7>:     push   %rbx
   0x00000000000026d3 <+8>:     sub    $0x50,%rsp
   0x00000000000026d7 <+12>:    lea    0x30(%rsp),%r14
   0x00000000000026dc <+17>:    mov    %r14,%rsi
   0x00000000000026df <+20>:    call   0x2b8f <read_six_numbers>
   0x00000000000026e4 <+25>:    mov    $0x1,%r13d
   0x00000000000026ea <+31>:    mov    %r14,%r12
   0x00000000000026ed <+34>:    jmp    0x279f <phase_6+212>
   0x00000000000026f2 <+39>:    call   0x2b53 <explode_bomb>
   0x00000000000026f7 <+44>:    cmp    $0x5,%r13d
   0x00000000000026fb <+48>:    jle    0x27b7 <phase_6+236>
   0x0000000000002701 <+54>:    mov    $0x0,%esi
   0x0000000000002706 <+59>:    mov    0x30(%rsp,%rsi,4),%ecx
   0x000000000000270a <+63>:    mov    $0x1,%eax
   0x000000000000270f <+68>:    lea    0x3bea(%rip),%rdx        # 0x6300 <node1>
   0x0000000000002716 <+75>:    cmp    $0x1,%ecx
   0x0000000000002719 <+78>:    jle    0x2726 <phase_6+91>
   0x000000000000271b <+80>:    mov    0x8(%rdx),%rdx
   0x000000000000271f <+84>:    add    $0x1,%eax
   0x0000000000002722 <+87>:    cmp    %ecx,%eax
   0x0000000000002724 <+89>:    jne    0x271b <phase_6+80>
   0x0000000000002726 <+91>:    mov    %rdx,(%rsp,%rsi,8)
   0x000000000000272a <+95>:    add    $0x1,%rsi
   0x000000000000272e <+99>:    cmp    $0x6,%rsi
   0x0000000000002732 <+103>:   jne    0x2706 <phase_6+59>
   0x0000000000002734 <+105>:   mov    (%rsp),%rbx
   0x0000000000002738 <+109>:   mov    0x8(%rsp),%rax
   0x000000000000273d <+114>:   mov    %rax,0x8(%rbx)
   0x0000000000002741 <+118>:   mov    0x10(%rsp),%rdx
   0x0000000000002746 <+123>:   mov    %rdx,0x8(%rax)
   0x000000000000274a <+127>:   mov    0x18(%rsp),%rax
   0x000000000000274f <+132>:   mov    %rax,0x8(%rdx)
   0x0000000000002753 <+136>:   mov    0x20(%rsp),%rdx
   0x0000000000002758 <+141>:   mov    %rdx,0x8(%rax)
   0x000000000000275c <+145>:   mov    0x28(%rsp),%rax
   0x0000000000002761 <+150>:   mov    %rax,0x8(%rdx)
   0x0000000000002765 <+154>:   movq   $0x0,0x8(%rax)
   0x000000000000276d <+162>:   mov    $0x5,%ebp
   0x0000000000002772 <+167>:   jmp    0x27c5 <phase_6+250>
```





Le résultat attendue de la phase_6 est une liste de 6 nombre , chaque chiffre doit être inférieur à 6 sinon la bombe explose. De plus on doit vérifier que chaque valeur soit supérieur au précédent sinon la bombe explosera aussi. Maintenant analysons les valeur du tableau:

* Node 1= 0x00000247 = 583
* Node 2= 0x000001dc = 476
* Node 3= 0x0000015b= 347
* Node 4= 0x0000010e = 270
* Node 5= 0x00000119 = 281

Ainsi dans en triant le tableau on aura : **270,281,347,476,583 (soit 4,5,3,2,1)**

Ainsi le résultat attendu de la phase_6 est: **4,5,3,2,1,6.**

Resultat final:

```
(gdb) r
Starting program: /info/nouveaux/akbas/bomb99/bomb
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
For NASA, space is still a high priority.
Phase 1 defused. How about the next one?
1 2 4 8 16 32
That's number 2.  Keep going!
2 i 276
Halfway there!
40 2
So you got that one.  Try this one.
BCCEEF
Good work!  On to the next...
4 5 3 2 1 6
Congratulations! You've defused the bomb!
Your instructor has been notified and will verify your solution.
[Inferior 1 (process 4053319) exited normally]
```

Chaque phase avait son niveau de difficulté, pour ma part la phase_3 m'a pris plus de temps à trouver la solutions (14 heure). Cependant c'était satisfaisant de trouvé chaque phase. Au total j'ai mis 30 heures pour finir ce TP.