```
/*
 * gcc −o stack −g stack.c
 * gcc −S −o stack.s stack.c
 * gdb stack
 * try the following gdb commands:
 * list sumNums
 * break 88
 * run
 * print i
 * step
 * print i
 * info registers
 * display i
 * print &sum
 * print &main
 * x/6w 0xbffff9e0
 * disassemble main
 * frame 0
 * frame 1
 * set {int}0xbffffa8c = &hacked
 */


/*
Memory looks like (1 byte=8 bits at a time)
Note: This is a 32 bit architecture, so I show
4 bytes in a line. The machine likes to fetch
memory 4 bytes at a time on 4 byte boundaries.
Larger scalars (>1 byte) are 4 byte boundary alligned.

Note: Change in organization of memory from our declarations

      Address   +0     +1     +2     +3       Whats there

      00000000
      00000004
      ...
      globals are in this part of memory
      08049698   f      f      f      f
      0804969c   l      l      l      l
      080496a0   i      i      i      i
      080496a4   c      ?      ?      ?
      080496a8   d      d      d      d
      080496ac   d      d      d      d
      080496b0   s      s      ?      ?
      ...
      the stack is down here and grows up
      ...
esp−> bffff990   c      c      c      c  esp 0xbffff9a0 (top of stack is here)
      bffff994   c      c      c      c
      bffff998   c      c      c      c
      bffff99c   c      c      c      c
      bffff9a0   ?      ?      ?      ?
      bffff9a4   ?      ?      ?      ?
      bffff9a8 sum    sum    sum    sum
      bffff9ac   i      i      i      i  (locals on the stack in reverse order)
      bffff9b0   ?      ?      ?      ?
      bffff9b4   ?      ?      ?      ?
ebp−> bffff9b8 sfp    sfp    sfp    sfp  (this is the old frame pointer old ebp)
      bffff9bc ret    ret    ret    ret  (you can see this by x/24w 0xbffff9ac a
nd disassemble main)
      bffff9d0   n      n      n      n
      bffff9d4   m      m      m      m  (args pushed on stack in reverse order)

      ...
      fffffffc   ?      ?      ?      ?

Symbol     address relative to ebp (see stack.s or disassemble sumNums)
   c              −40(%ebp)
  sum             −16(%ebp)
   i              −12(%ebp)
   n                8(%ebp)
   m               12(%ebp)

*/

#include<stdio.h>

// Global variables, available to the whole program
char c;
short s;
int i;
long l;
float f;
double d;

int sumNums(int n, int m){
    int i;
    int sum=0;
    char c[16];
    i=n;
    while(i<m){
        sum=sum+i;
        i=i+1;
    }
    c[0]=0x41;
    c[1]=0x41;
    c[2]=0x41;
    c[3]=0x41;
    return sum;
}
void f1(int a, int b, int c, int d){

}
void hacked(){
    printf("I've been hacked\n");
}
int main(int argc, char ** argv){
    sumNums(3,7);
    f1(1,2,3,4);
    f1(1,2,3,4);
}
    .file   "stack.c"
    .version    "01.01"
gcc2_compiled.:
.text
    .align 4
.globl sumNums
    .type    sumNums,@function
sumNums:
    pushl   %ebp
    movl    %esp, %ebp
    subl    $40, %esp
    movl    $0, −16(%ebp)
    movl    8(%ebp), %eax
    movl    %eax, −12(%ebp)
    .p2align 2
.L3:
    movl    −12(%ebp), %eax
    cmpl    12(%ebp), %eax
    jl  .L5
```

```
    jmp .L4
    .p2align 2
.L5:
    movl    -12(%ebp), %eax
    leal    -16(%ebp), %edx
    addl    %eax, (%edx)
    leal    -12(%ebp), %eax
    incl    (%eax)
    jmp .L3
    .p2align 2
.L4:
    movl    -16(%ebp), %eax
    movl    %eax, %eax
    leave
    ret
.Lfe1:
    .size    sumNums,.Lfe1-sumNums
    .align 4
.globl f1
    .type    f1,@function
f1:
    pushl   %ebp
    movl    %esp, %ebp
    popl    %ebp
    ret
.Lfe2:
    .size    f1,.Lfe2-f1
        .section    .rodata
.LC0:
    .string "I've been hacked\n"
.text
    .align 4
.globl hacked
    .type    hacked,@function
hacked:
    pushl   %ebp
    movl    %esp, %ebp
    subl    $8, %esp
    subl    $12, %esp
    pushl   $.LC0
    call    printf
    addl    $16, %esp
    leave
    ret
.Lfe3:
    .size    hacked,.Lfe3-hacked
    .align 4
.globl main
    .type    main,@function
main:
    pushl   %ebp
    movl    %esp, %ebp
    subl    $8, %esp
    subl    $8, %esp
    pushl   $7
    pushl   $3
    call    sumNums
    addl    $16, %esp
    pushl   $4
    pushl   $3
    pushl   $2
    pushl   $1
    call    f1
    addl    $16, %esp
    pushl   $4
    pushl   $3
    pushl   $2
    pushl   $1
    call    f1
    addl    $16, %esp
    leave
    ret
.Lfe4:
    .size    main,.Lfe4-main
    .comm   c,1,1
    .comm   s,2,2
    .comm   i,4,4
    .comm   l,4,4
    .comm   f,4,4
    .comm   d,8,8
    .ident  "GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.1 2.96-98)"
#include<stdio.h>

void hacked(){
    printf("Hacked\n");
}
void f(){
    char c[1]; // actually don't need this, but ok.
    int *ref;
    int i;
    /*
     * You can use gdb to find the return address (to main)
     * on the stack. Change it so that it points
     * to hacked() instead. The result is that
     * instead of returning to main, this function
     * will return to hacked.
     */

    for(i=0;i<20;i++)c[i]='A'; // look for 0x41 on the stack

    // ref=(int *)(c+16);
    // printf("%x\n",ref);
    // *ref=&hacked;

}
int main(int argc, char ** argv){
    f();
}
```