This has a more elaborate program:

```
int A[10];

int g(int n) {
    return n*(n+1)/2;
}

void main(void) {
     int x;
     int t;
     x = 0;
     while (x < 10) {
         t = g(x);
         A[x] = t;
         write(x);
         write( "==>" );
         write(t);
         writeln();
         x = x + 1;
    }
}
```

#######################################

This declares the global array

```
.comm A, 80, 32
.section   .rodata
.WriteIntString: .string "%d "
.WritelnString: .string "\n"
.WriteStringString: .string "%s "
.ReadIntString: .string "%d"
.ArrayOverflowString: .string "You fell off the end of an array.\n"
.S0: .string "==>"
```

The string used in main( )

```
.text
.globl main
g:
```

Enter function g

```
       movq %rsp, %rbx            #set up the frame pointer
        sub $0, %rsp            #allocate local variables
       movq 16(%rbx), %rax              #param value
       push %rax                #saving the left operand on the stack
```

n

```
       movq 16(%rbx), %rax              #param value
       push %rax                #saving the left operand on the stack
       movl $1, %eax            #putting value into ac
       addl 0(%rsp), %eax             #performing addition
```

n+1

```
       addq $8, %rsp            #popping the value saved on the stack
       imul 0(%rsp), %eax             #performing multiplication
```

n*(n+1)

```
       addq $8, %rsp            #popping the value saved on the stack
       push %rax                #saving the left operand on the stack
```

Division by 2

```
       movl $2, %eax            #putting value into ac
       movl %eax, %ebx            #moving divisor to rbx
       movl 0(%rsp), %eax             #moving dividend to ac
       cltq                       #sign-extend the upper half of rax
       cqto                       #sign-extending rax into rdx
       idivl %ebx                 #performing division
```

```
        addq $8, %rsp            #popping the value saved on the stack
         add $0, %rsp            #deallocate local variables    return n*(n+1)/2
         ret                      #return from the function
         add $0, %rsp            #deallocate local variables    return from g
         ret                      #return from the function
main:
        movq %rsp, %rbx            #set up the frame pointer    enter main( )
         sub $16, %rsp            #allocate local variables
        movq %rbx, %rax            #put the frame pointer into ac
         sub $8, %rax            #local variable address
        push %rax                #push the lvalue
        movl $0, %eax            #putting value into ac         x=0
        movq 0(%rsp), %rsi            #put the lvalue into rsi
        movq %rax, 0(%rsi)            #assign
        addq $8, %rsp            #pop the lvalue from the stack
.L0:
        movq -8(%rbx), %rax            #local variable value
        push %rax                #saving the left operand on the stack
        movl $10, %eax            #putting value into ac
        cmpl %eax, 0(%rsp)            #comparing operands       x < 10
        jl .L2                      #go to true branch
        movl $0, %eax            #not less: ac=0
        jmp .L3                      #jump overtrue branch
.L2:
        movl $1, %eax            #less: ac=1
.L3:
        addq $8, %rsp            #popping the value saved on the stack
        cmpl $0, %eax            #test if ac is 0                if "x<10" is false
        je .L1                      #WHILE: on true jump out of loop   leave the loop
        movq %rbx, %rax            #put the frame pointer into ac
         sub $16, %rax            #local variable address
        push %rax                #push the lvalue               L-value of t
        movq -8(%rbx), %rax      #local variable value
        push %rax                #pushing argument
        push %rbx                #pushing the frame pointer      call g(x)
        call g                      #calling the function
         pop %rbx                #retrieving the frame pointer
         add $8, %rsp            #removing args from the stack
        movq 0(%rsp), %rsi            #put the lvalue into rsi
        movq %rax, 0(%rsi)            #assign                   t = g(x)
        addq $8, %rsp            #pop the lvalue from the stack
        movq $A, %rax            #global array value
        push %rax                #save array address on the stack   minimal
        movq -8(%rbx), %rax            #local variable value        bounds
        cmpl $0, %eax            #check for negative bounds          checking
        jge .L4                      #jump over exit if okay
        movq $.ArrayOverflowString, %rdi            #printf string = arg1
        movl $0, %eax            #clear the return value
        call printf                        #call the C-lib printf function
        movl $1, %eax            #system exit
        movl $1, %ebx            #error code
        call syscall                        #halt
.L4:
        imul $8, %rax            #convert index to byte offset   L-value of A[x]
```

```
         add 0(%rsp), %rax             #compute address of element
         add $8, %rsp             #remove array address from stack
        push %rax                 #push the lvalue
        movq -16(%rbx), %rax            #local variable value          t
        movq 0(%rsp), %rsi            #put the lvalue into rsi         A[x]=t
        movq %rax, 0(%rsi)            #assign
        addq $8, %rsp             #pop the lvalue from the stack
        movq -8(%rbx), %rax            #local variable value
        movl %eax, %esi             #value to print = arg2             write(x)
        movq $.WriteIntString, %rdi            #printf string = arg1
        movl $0, %eax             #clear the return value
        call printf                       #call the C-lib printf function
        movq $.S0, %rax             #putting string value into ac
        movq %rax, %rsi             #string to print = arg2
        movq $.WriteStringString, %rdi            #printf string = arg1   write("==>")
        movl $0, %eax             #clear the return value
        call printf                       #call the C-lib printf function
        movq -16(%rbx), %rax            #local variable value
        movl %eax, %esi             #value to print = arg2             write(t)
        movq $.WriteIntString, %rdi            #printf string = arg1
        movl $0, %eax             #clear the return value
        call printf                       #call the C-lib printf function
        movq $.WritelnString, %rdi            #printf string = arg1    writeln( )
        movl $0, %eax             #clear the return value
        call printf                       #call the C-lib printf function
        movq %rbx, %rax             #put the frame pointer into ac
         sub $8, %rax             #local variable address
        push %rax                 #push the lvalue
        movq -8(%rbx), %rax            #local variable value
        push %rax                 #saving the left operand on the stack    x = x+1
        movl $1, %eax             #putting value into ac
        addl 0(%rsp), %eax            #performing addition
        addq $8, %rsp             #popping the value saved on the stack
        movq 0(%rsp), %rsi            #put the lvalue into rsi
        movq %rax, 0(%rsi)            #assign
        addq $8, %rsp             #pop the lvalue from the stack
        jmp .L0                      #WHILE: jump back to top        go back to the top of the loop
.L1:
         add $16, %rsp             #deallocate local variables
         ret                       #return from the function         return from main( )
```