- 指针也可以指向函数。这提供了一个很强大的存储和向代码传递引用的功能，这些引用可以被程序的某个其他部分调用。例如，如果我们有一个函数，用下面这个原型定义：

```
int fun(int x, int *p);
```

然后，我们可以声明一个指针 fp，将它赋值为这个函数，代码如下：

```
int (*fp)(int, int *);
fp  = fun;
```

然后用这个指针来调用这个函数：

```
int y = 1;
int result = fp(3, &y);
```

函数指针的值是该函数机器代码表示中第一条指令的地址。

int a[3]
*a[3]
int (*a)[3]
int *(a[3])

给 C 语言初学者　函数指针

函数指针声明的语法对程序员新手来说特别难以理解。对于以下声明：

```
int (*f)(int*);
```

要从里(从“f”开始)往外读。因此，我们看到像“(*f)”表明的那样，f 是一个指针；而“(*f)(int*)”表明 f 是一个指向函数的指针，这个函数以一个 int* 作为参数。最后，我们看到，它是指向以 int * 为参数并返回 int 的函数的指针。

*f 两边的括号是必需的，否则声明变成

```
int *f(int*);
```

它会被解读成

```
(int *) f(int*);
```

也就是说，它会被解释成一个函数原型，声明了一个函数 f，它以一个 int * 作为参数并返回一个 int* 。

Kernighan 和 Ritchie [61，5.12 节]提供了一个有关阅读 C 声明的很有帮助的教程。

3.10.2 应用：使用 GDB 调试器

GNU 的调试器 GDB 提供了许多有用的特性，支持机器级程序的运行时评估和分析。对于本书中的示例和练习，我们试图通过阅读代码，来推断出程序的行为。有了 GDB，可以观察正在运行的程序，同时又对程序的执行有相当的控制，这使得研究程序的行为变为可能。

图 3-39 给出了一些 GDB 命令的例子，帮助研究机器级 x86-64 程序。先运行 OBJDUMP 来获得程序的反汇编版本，是很有好处的。我们的示例都基于对文件 prog 运行 GDB，程序的描述和反汇编见 3.2.3 节。我们用下面的命令行来启动 GDB：

```
linux> gdb prog
```

通常的方法是在程序中感兴趣的地方附近设置断点。断点可以设置在函数入口后面，或是一个程序的地址处。程序在执行过程中遇到一个断点时，程序会停下来，并将控制返回给用户。在断点处，我们能够以各种方式查看各个寄存器和内存位置。我们也可以单步跟踪程序，一次只执行几条指令，或是前进到下一个断点。

命令	效果
开始和停止	
quit	退出 GDB
run	运行程序(在此给出命令行参数)
kill	停止程序
断点	
break multstore	在函数 multstore 入口处设置断点
break * 0x400540	在地址 0x400540 处设置断点
delete 1	删除断点 1
delete	删除所有断点
执行	
stepi	执行 1 条指令
stepi 4	执行 4 条指令
nexti	类似于 stepi，但以函数调用为单位
continue	继续执行
finish	运行到当前函数返回
检查代码	
disas	反汇编当前函数
disas multstore	反汇编函数 multstore
disas 0x400544	反汇编位于地址 0x400544 附近的函数
disas 0x400540,0x40054d	反汇编指定地址范围内的代码
print /x $rip	以十六进制输出程序计数器的值
检查数据	
print $rax	以十进制输出%rax 的内容
print /x $rax	以十六进制输出%rax 的内容
print /t $rax	以二进制输出%rax 的内容
print 0x100	输出 0x100 的十进制表示
print /x 555	输出 555 的十六进制表示
print /x ($rsp+ 8)	以十六进制输出%rsp 的内容加上 8
print *(long *) 0x7fffffffe818	输出位于地址 0x7fffffffe818 的长整数
print *(long *)($rsp+ 8)	输出位于地址%rsp+8 处的长整数
x/2g 0x7fffffffe818	检查从地址 0x7fffffffe818 开始的双(8 字节)字
x/20bmultstore	检查函数 multstore 的前 20 个字节
有用的信息	
info frame	有关当前栈帧的信息
info registers	所有寄存器的值
help	获取有关 GDB 的信息

图 3-39　GDB 命令示例。说明了一些 GDB 支持机器级程序调试的方式

正如我们的示例表明的那样，GDB 的命令语法有点晦涩，但是在线帮助信息(用 GDB 的 help 命令调用)能克服这些毛病。相对于使用命令行接口来访问 GDB，许多程序员更愿意使用 DDD，它是 GDB 的一个扩展，提供了图形用户界面。

3.10.3　内存越界引用和缓冲区溢出

我们已经看到，C 对于数组引用不进行任何边界检查，而且局部变量和状态信息(例如保存的寄存器值和返回地址)都存放在栈中。这两种情况结合到一起就能导致严重的程序错误，对越界的数组元素的写操作会破坏存储在栈中的状态信息。当程序使用这个被破