1. Linux 安装和配置

ii.

```
noob@noob-virtual-machine:~$ vim --version
VIM - Vi IMproved 7.4 (2013 Aug 10, compiled Nov 24 2016 16:44:48)
Included patches: 1-1689
Extra patches: 8.0.0056
Modified by pkg-vim-maintainers@lists.alioth.debian.org
Compiled by pkg-vim-maintainers@lists.alioth.debian.org
```

```
noob@noob-virtual-machine:~$ git --version
git version 2.7.4
noob@noob-virtual-machine:~$ gcc --version
gcc (Ubuntu 5.4.0-6ubuntu1~16.04.10) 5.4.0 20160609
Copyright (C) 2015 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

noob@noob-virtual-machine:~$ as --version
GNU assembler (GNU Binutils for Ubuntu) 2.26.1
Copyright (C) 2015 Free Software Foundation, Inc.
This program is free software; you may redistribute it under the terms of
the GNU General Public License version 3 or later.
This program has absolutely no warranty.
This assembler was configured for a target of `i686-linux-gnu'.
noob@noob-virtual-machine:~$ objdump --version
GNU objdump (GNU Binutils for Ubuntu) 2.26.1
Copyright (C) 2015 Free Software Foundation, Inc.
This program is free software; you may redistribute it under the terms of
the GNU General Public License version 3 or (at your option) any later version.
This program has absolutely no warranty.
noob@noob-virtual-machine:~$ gdb --version
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.04) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word".
```

3.熟悉工具

心型截图如下:

```
noob@noob-virtual-machine:~/workspace/lab01$ ./heart
 *    *
*******
 *****
   *
161130118
```

i.

```
 1a:    0a 31
 1c:    36 31 31
 1f:    33 30
 21:    31 31
 23:    38 0a
```

两个 0x0a 之间的十六进制数字，从左到右，从上到下，即为我的学号.
即 0x31, 0x36, 0x31, 0x31, 0x33, 0x30, 0x31, 0x31, 0x38.

ii.

```
noob@noob-virtual-machine:~/workspace/lab01$ gcc -E hello.c -o hello.i
noob@noob-virtual-machine:~/workspace/lab01$ gcc -S hello.i -o hello.s
noob@noob-virtual-machine:~/workspace/lab01$ gcc -c hello.s -o hello.o
noob@noob-virtual-machine:~/workspace/lab01$ gcc -o hello hello.o
noob@noob-virtual-machine:~/workspace/lab01$ ls
dog  heart  heart.o  hello  hello.c  hello.i  hello.o  hello.s
noob@noob-virtual-machine:~/workspace/lab01$ ./hello
hello, world!
```

4.数据的表示范围及不同类型的数据长度实验

i.

```
noob@noob-virtual-machine:~/workspace/lab01$ ./sqr
The 40000*40000 is 1600000000
The 50000*50000 is -1794967296
```

32 位操作系统中 int 类型占四个字节，其表示范围为-2147483648～2147483647. 而 50000*50000 实际结果为 2500000000，超过 int 类型能表示的范围，产生溢出现象. 由于计算机底层用二进制补码表示数字，且在 int 类型下 2500000000 的二进制补码和-1794967296 的二进制补码相同，又由于 int 类型下该补码只有一个合法的解释，即-1794967296，所以该程序产生如图所示的现象.

ii.

46340.

iii.

采用 unsigned int 来保存变量 i, j.

```
noob@noob-virtual-machine:~/workspace/lab01$ cat sqr.c
#include<stdio.h>

int main()
{
    unsigned int i, j;
    i = 40000;
    j = i * i;
    printf ("The 40000*40000 is %u\n", j);
    i = 50000;
    j = i * i;
    printf ("The 50000*50000 is %u\n", j);
    return 0;
}
```

5.矩阵运算执行时间比较

i.

```
noob@noob-virtual-machine:~/workspace/lab01$ ./matrix
copyij 0.023162 s
copyji 0.187039 s
noob@noob-virtual-machine:~/workspace/lab01$ ./matrix
copyij 0.020142 s
copyji 0.203091 s
noob@noob-virtual-machine:~/workspace/lab01$ ./matrix
copyij 0.032671 s
copyji 0.187554 s
noob@noob-virtual-machine:~/workspace/lab01$ ./matrix
copyij 0.029708 s
copyji 0.183917 s
noob@noob-virtual-machine:~/workspace/lab01$ ./matrix
copyij 0.028694 s
copyji 0.191094 s
```

ii.

CPU 读取内存某地址处的值，并不是每次都从内存中取值，有时候会从 cache 里读取. 当初次访问数组的时候，CPU 会把连续一块内存地址上的值都读到 cache 里，后续 CPU 接受到一个内存地址要读取数据时，会先在 cache 里寻找，若没有再去内存中取.

在 C 语言的底层实现中，二维数组是按行优先的顺序，将数组中的数据存到一块连续内存上的. 因此以行优先的方式(copyij)进行访问时，会相应提高缓存命中率，且 cache 上的读写速度要明显高于内存上的读写速度，所以以行优先方式进行访问时的速度明显优于以列优先方式进行访问时的速度.