中国科学技术大学

程序设计课程回顾与UML简介

徐 伟

E-mail: xuweihf@ustc.edu.cn

课程概述

- 程序设计回顾
- 常用编译工具使用
- 程序如何运行起来?
- 流程图与常用UML图

C语言

排名	编程语言	流行度	对比上月	年度明星
1	Python	21.81%	▼ -0.8%	2024, 2021
2	C	11.05%	▲ 0.06%	2019, 2017
3	C++	8.55%	▼ -0.12%	2022, 2003
4	Java	8.12%	▼ -0.59%	2015, 2005
5	C#	6.83%	▼ -0.56%	2025, 2023
6	JavaScript	2.92%	▼ -0.11%	2014

语言规范标准

当前主流：C17/C11

最新标准：C23

（ISO/IEC 9899：2024）

行业编码规范

MISRA C：工业界影响最广的C语言编码规范

安全监管要求：内存安全带来的新趋势

作为内存安全语言的代表，Rust已经开始渗透到C语言的传统领域

C语言基本结构

```c
#include <stdio.h>

int main() {
    printf("Hello, World!\n");
    return 0;
}
```

代码组成结构如下：

预处理器指令：如 #include 和 #define。

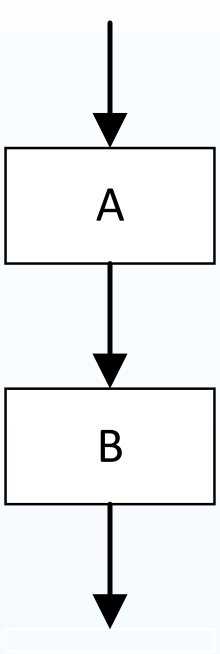
主函数：每个 C 程序都有一个 main() 函数。

变量声明：声明程序中使用的变量。

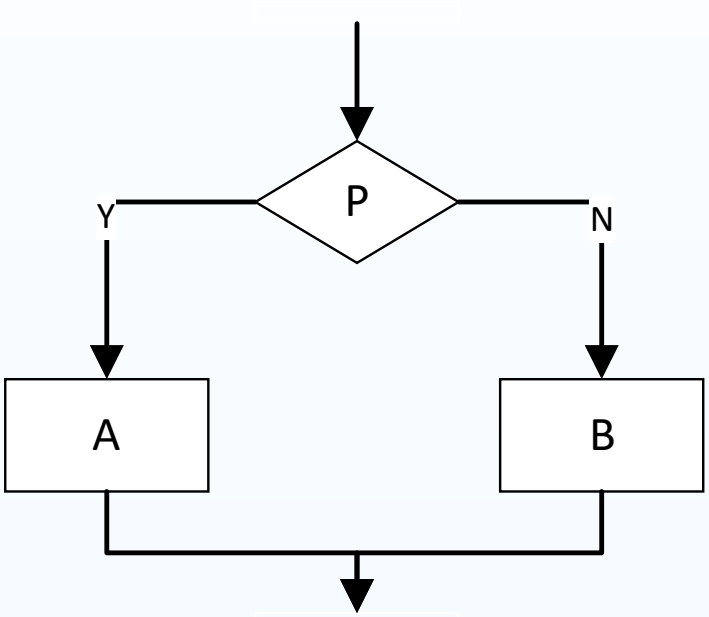
函数定义：定义程序中使用的函数。

程序设计基本结构

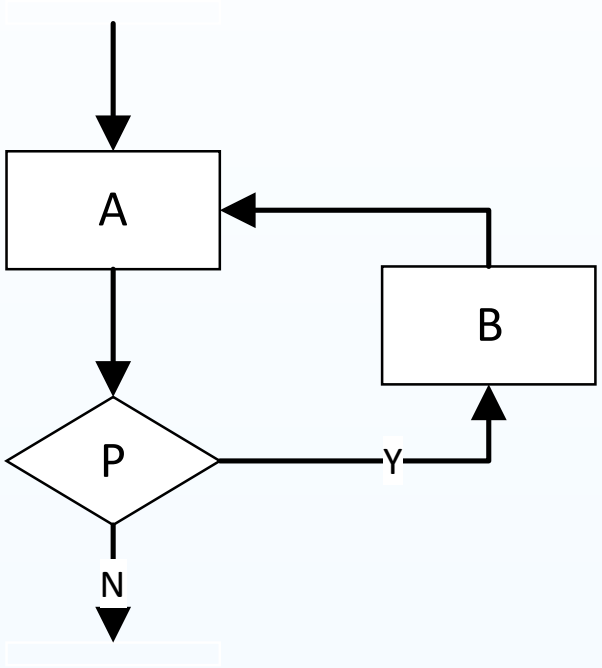
顺序结构



选择（分支）结构



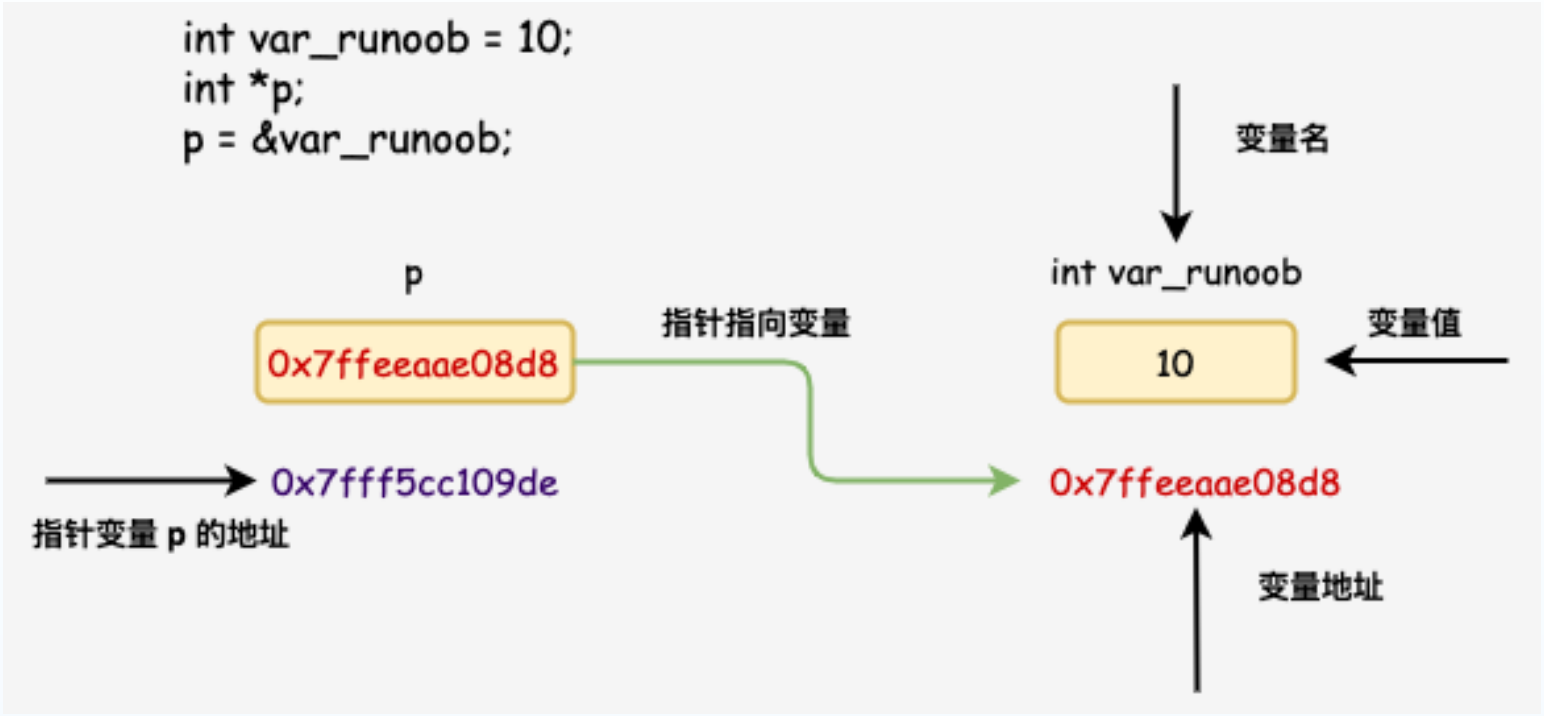
循环结构



指针

```
#include <stdio.h>
int main ()
{
   int var_runoob = 10;
   int *p; // 定义指针变量
   p = &var_runoob;
   printf("var_runoob 变量的地址： %p\n", p);
   return 0; }
```

指针



运行结果

var_runoob 变量的地址： 0x7ffeeaae08d8

指针

```
#include <stdio.h>
int main ()
{
    int *p = NULL;
    printf(“p的地址：  %p\n", p);
    return 0; }
```

指针

```
#include <stdio.h>
int main ()
{
   int *p = NULL;
   printf("p的地址：  %p\n", p);
   return 0; }
```

运行结果
p的地址：0x0

大多数的操作系统上，不允许访问地址为 0 内存（该内存是操作系统保留）
内存地址 0 有特别意义，表明该指针不指向一个可访问的内存

检查指针空指针语句：if(p) / if(!p)

课程概述

- 程序设计回顾
- 常用编译工具使用
- 程序如何运行起来?
- 流程图与常用UML图

程序的开发与编译

- Linux下的软件开发
  - gcc

    gcc（GNU Compiler Collection，GNU编译器套件）

    GNU开发的编程语言编译器

    现已被大多数类Unix操作系统（如Linux、BSD、Mac OS X等）采纳为标准的编译器，在微软的Windows上也可以使用gcc

  - gdb

    gdb （ GNU Project Debugger ， GNU项目调试器）

    小巧而强大，尺有所短，寸有所长

程序的开发与编译

- Linux下的软件开发
  - gcc安装

    sudo apt-get update

    sudo apt-get install gcc

  - gdb安装

    sudo apt-get update

    sudo apt-get install gdb

示例运行环境 Ubuntu 22.04.1

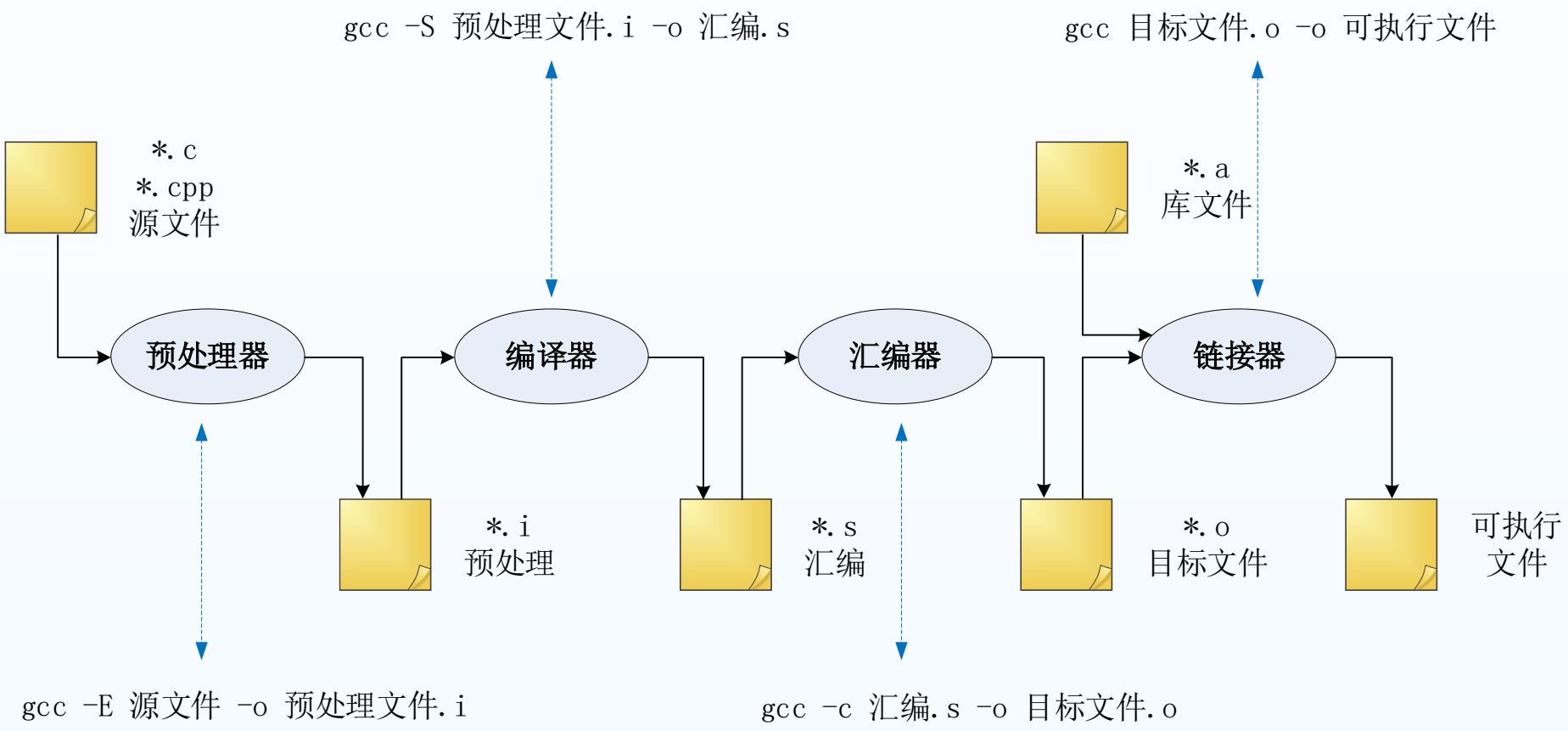
gcc 7.5.0　　gdb 8.1.1

程序的开发与编译

gcc/g++选项	功能	目标文件常用后缀
-E（大写）	预处理指定的源文件，不进行编译。	.i
-S（大写）	编译指定的源文件，生成 汇编文件，但是不进行汇编。	.s
-c	编译、汇编指定的源文件，生成目标文件，但是不进行链接。	.o
-o	指定生成目标文件的文件名	
-m32	按照32位进行编译	
-m64	按照64位进行编译	

有关更多编译指令，可自行查看GCC手册

程序的开发与编译


gcc -S 预处理文件.i -o 汇编.s
gcc 目标文件.o -o 可执行文件
*.c
*.cpp
源文件
*.a
库文件
预处理器
编译器
汇编器
链接器
*.i
预处理
*.s
汇编
*.o
目标文件
可执行
文件
gcc -E 源文件 -o 预处理文件.i
gcc -c 汇编.s -o 目标文件.o

程序的开发与编译

- 示例程序sample.c

```c
#include <stdio.h>

int main()
{
    printf("Hello, world!\n");
    return 0;
}
```

gcc –E sample.c –o sample.i

```c
# 1 "sample.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "/usr/include/stdc-predef.h" 1 3 4
```

```c
extern int fclose (FILE *__stream);
```

```c
extern int fprintf (FILE *__restrict __stream,
      const char *__restrict __format, ...);
```

```c
int main()
{
    printf("Hello, world!\n");
    return 0;
}
```

程序的开发与编译

- 示例程序sample.c

```c
#include <stdio.h>

int main()
{
    printf("Hello, world!\n");
    return 0;
}
```

gcc –S sample.i –o sample.s

```
        .file   "sample.c"
        .section        .rodata
.LC0:
        .string "Hello, world!"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        pushq   %rbp
        movq    %rsp, %rbp
        movl    $.LC0, %edi
        call    puts
        movl    $0, %eax
        popq    %rbp
        ret
```

程序的开发与编译

- 示例程序sample.c

```
1  #include <stdio.h>
2
3  int main()
4  {
5      printf("Hello, world!\n");
6      return 0;
7  }
```

gcc –c sample.s –o sample.o

gcc sample.o –o sample

```
root@evassh-4250186:/data/workspace/myshixun# gcc -c sample.s -o sample.o
root@evassh-4250186:/data/workspace/myshixun# ls
platform-script  sample.c  sample.i  sample.o  sample.s  secret
root@evassh-4250186:/data/workspace/myshixun# gcc sample.o -o sample
root@evassh-4250186:/data/workspace/myshixun# ls
platform-script  sample  sample.c  sample.i  sample.o  sample.s  secret
root@evassh-4250186:/data/workspace/myshixun# ./sample
Hello, world!
```

程序的开发与编译

优化等级	简要说明
-Ofast	在-O3级别的基础上，开启更多**激进优化项**，该优化等级不会严格遵循语言标准
-O3	在-O2级别的基础上，开启了更多的**高级优化项**，以编译时间、代码大小、内存为代价获取更高的性能。
-Os	在-O2级别的基础上，开启**降低生成代码体量**的优化
-O2	开启了大多数**中级优化**，会改善编译时间开销和最终生成代码性能
-O/-O1	优化效果介于-O0和-O2之间
-O0	默认优化等级，即**不开启编译优化**，只尝试减少编译时间

举例——优化对代码性能的影响

```
#include <stdio.h>
#include <time.h>
int main() {
    int loop = 1000000000;
    long sum = 0;
    int start_time = clock();
    int index = 0;
    for (index = 0; index < loop; index ++)
    {
        sum += index;
    }
    int end_time = clock();
    printf("Sum : %ld, Time Cost : %lf \n", sum,
            (end_time - start_time) * 1.0 / CLOCKS_PER_SEC);
    return 0;
}
```

循环次数定义

开始计时

循环体

结束计时

代码运行时间输出

举例——优化对代码性能的影响

- gcc –O0 无优化执行

```
gloit@gloit-x1c ~/2022_compiler_demo  master gcc -O0 add.c
gloit@gloit-x1c ~/2022_compiler_demo  master ./a.out
Sum: 49999999500000000, Time Cost: 3.415244
```

- gcc –O1 中级优化执行

性能提升5倍

```
gloit@gloit-x1c ~/2022_compiler_demo  master gcc -O1 add.c
gloit@gloit-x1c ~/2022_compiler_demo  master ./a.out
Sum: 49999999500000000, Time Cost: 0.554717
```

- gcc –O2 高级优化执行

性能提升数十万倍

```
gloit@gloit-x1c ~/2022_compiler_demo  master gcc -O2 add.c
gloit@gloit-x1c ~/2022_compiler_demo  master ./a.out
Sum: 49999999500000000, Time Cost: 0.000002
```

gdb入门

- 例题5：poj2080日历问题（P130)

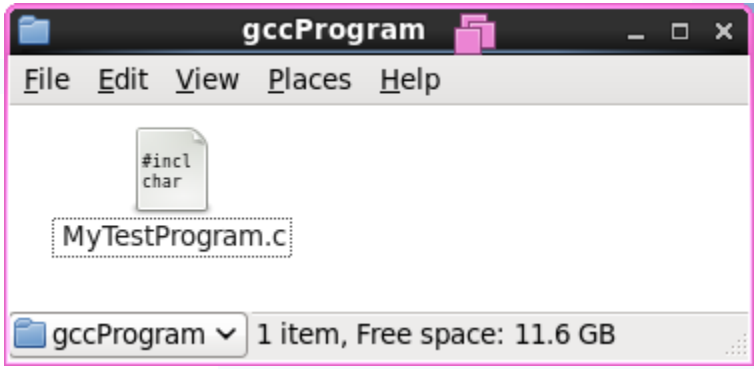
问题描述：

在我们现在使用的日历中，闰年被定义为能被4整除的年份，但是能被100整除，而不能被400整除的年是例外，它们不是闰年。例如：1700，1800，1900 和 2100 不是闰年，而1600，2000 和 2400是闰年。

给定从公元2000年1月1日开始逝去得天数，你的任务是给出这一天是哪年哪月哪日星期几。

gdb入门

```c
#include <stdio.h>
char week[7][10]={"Saturday", "Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday"};

int year[2]={365,366};
//year[0]表示非闰年的天数，year[1]表示闰年的天数。

int month[2][12]={31,28,31,30,31,30,31,31,30,31,30,31,
31,29,31,30,31,30,31,31,30,31,30,31};
//month[0]表示非闰年里每个月的天数，month[1]表示闰年里每个月的天数。

int type(int m){

    //判断第m年是否是闰年，是则返回1，否则返回0。
    if(m % 4 != 0 || (m % 100 == 0 && m % 400 != 0))        return  0;  //不是闰年
    else return 1;   // 是闰年

}
```

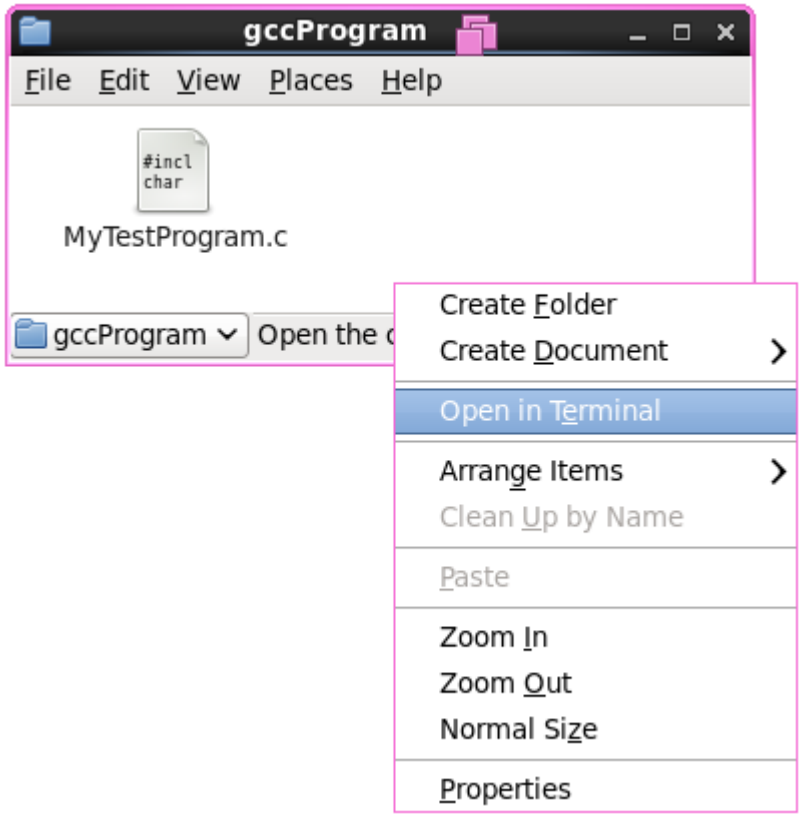



```c
int  main(void)
{
    int days, dayofweek;
 //days 表示输入的天数，dayofweek表示星期几。
    int i = 0, j = 0;
    while (scanf("%d", &days) && days != -1) {
        dayofweek = days % 7;
        for(i = 2000; days >= year[type(i)]; i++)
            days -= year[type(i)];
        for(j = 0; days >= month[ type(i) ][ j ]; j++)
            days -= month[ type(i) ][ j ];
        printf("%d-%02d-%02d %s\n",
            i, j + 1, days + 1, week[dayofweek]);
    }
    return 0;
}
```

gdb入门

- gcc文件编译
  - 直接编译运行

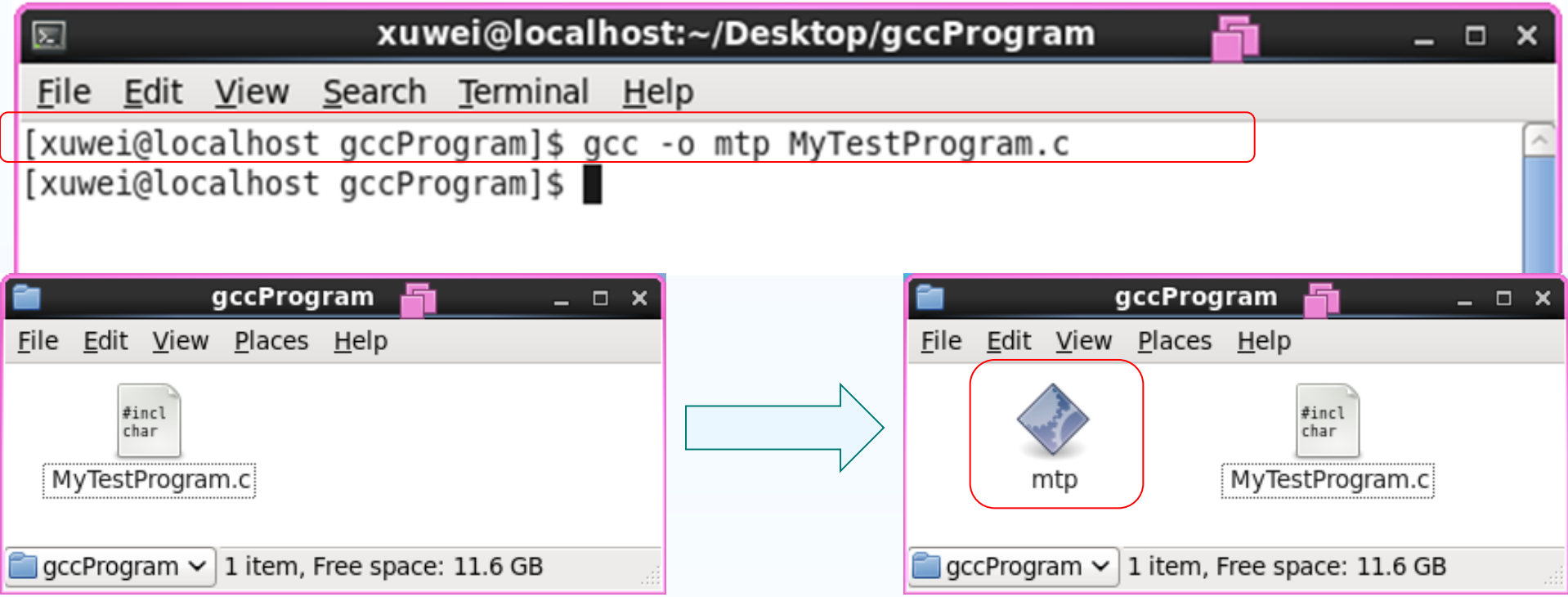
    gcc -o mtp MyTestProgram.c　生成可执行文件mtp



gdb入门

- gcc文件编译
  - 直接编译运行

    gcc -o mtp MyTestProgram.c　生成可执行文件mtp



gdb入门

- gcc文件编译
  - 需要gdb调试

    gcc -g MyTestProgram.c

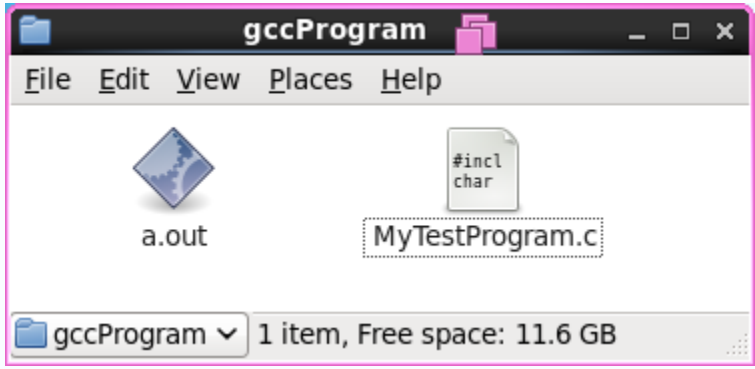
    默认生成a.out

    gcc -g -o mtp MyTestProgram.c

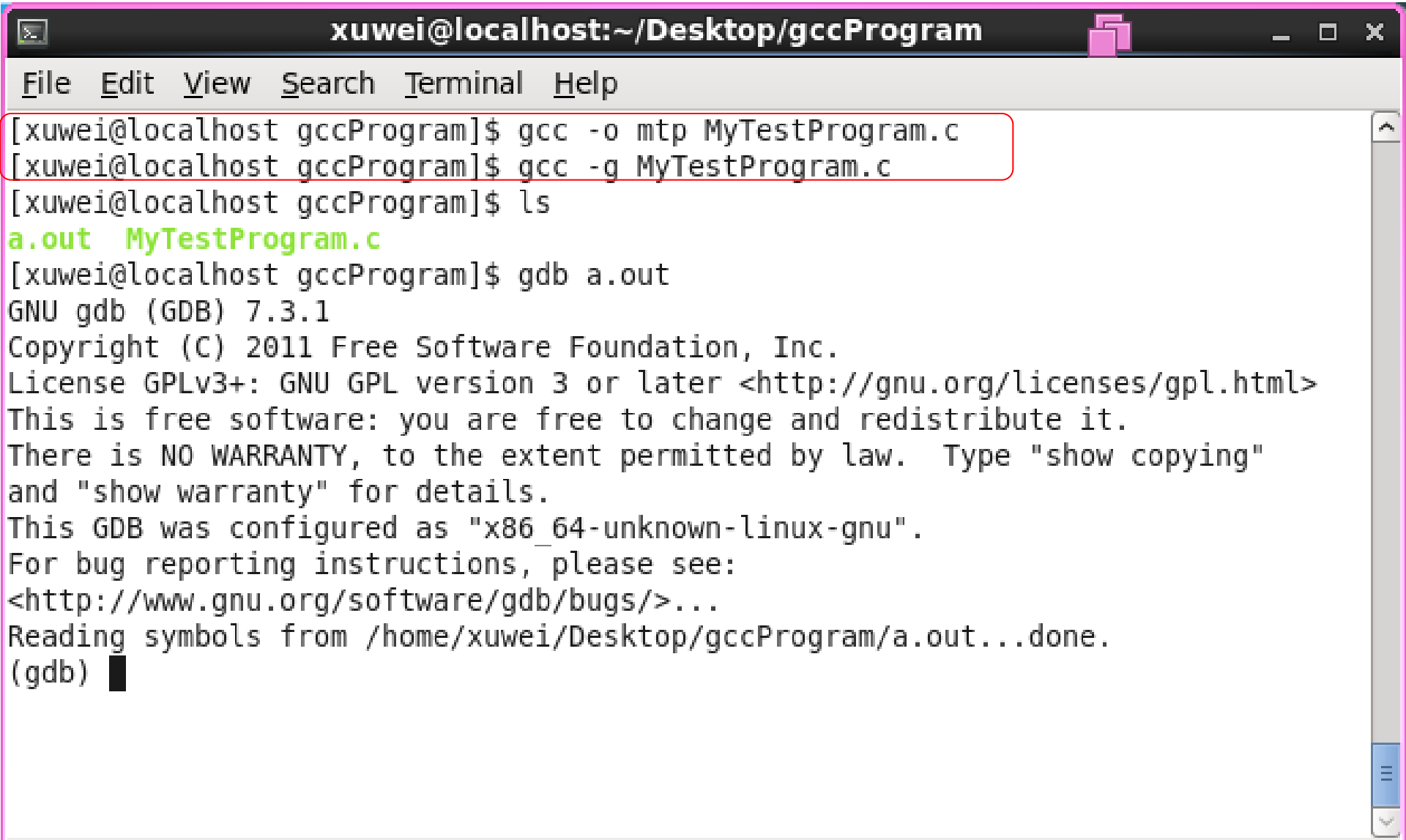
  - 终端运行 ./mtp　　./a.out

  - 较大工程，推荐使用makefile方式
  - C++程序采用g++命令进行编译

    g++ -g MyTestProgram.c



gdb入门


xuwei@localhost:~/Desktop/gccProgram
File Edit View Search Terminal Help
[xuwei@localhost gccProgram]$ gcc -o mtp MyTestProgram.c
[xuwei@localhost gccProgram]$ gcc -g MyTestProgram.c
[xuwei@localhost gccProgram]$ ls
a.out MyTestProgram.c
[xuwei@localhost gccProgram]$ gdb a.out
GNU gdb (GDB) 7.3.1
Copyright (C) 2011 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 "x86_64-unknown-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/xuwei/Desktop/gccProgram/a.out...done.
(gdb)

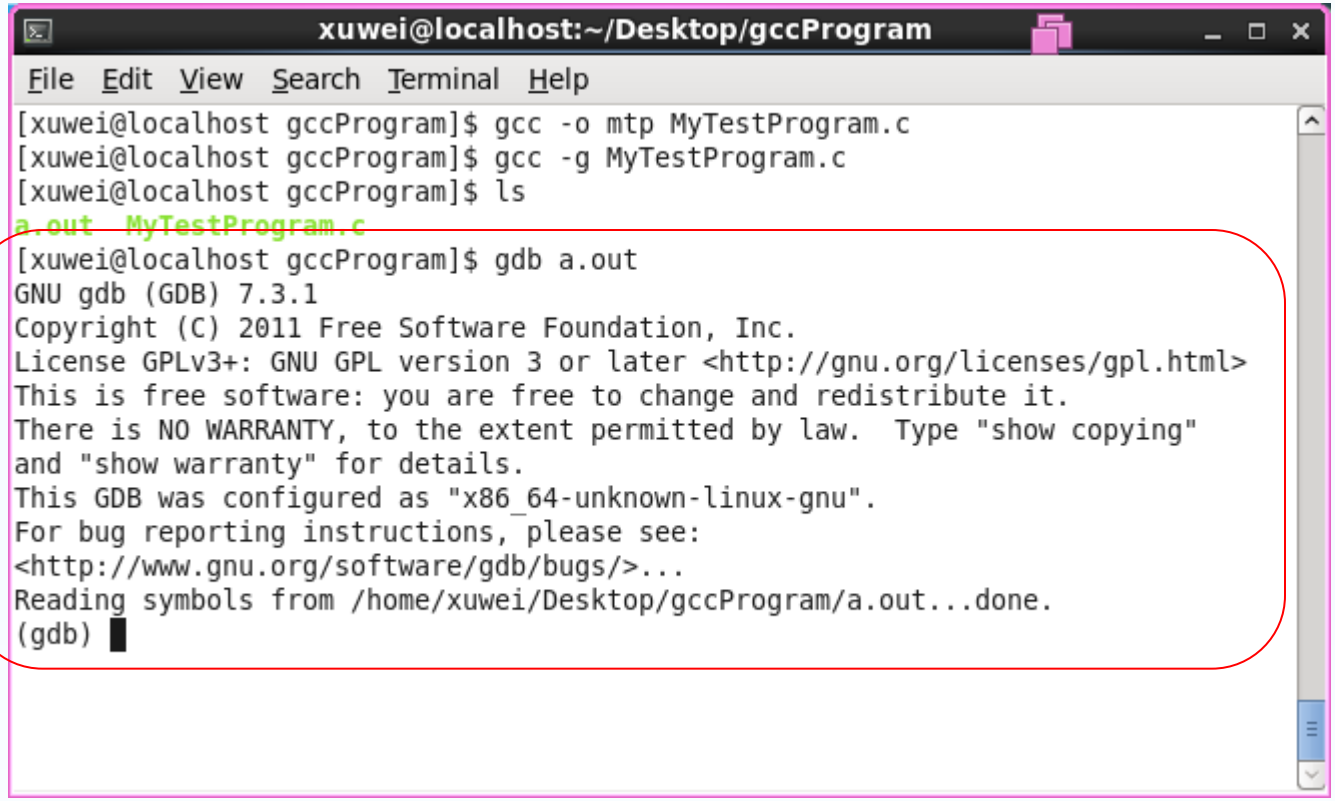
gdb

- gdb使用
  - gdb

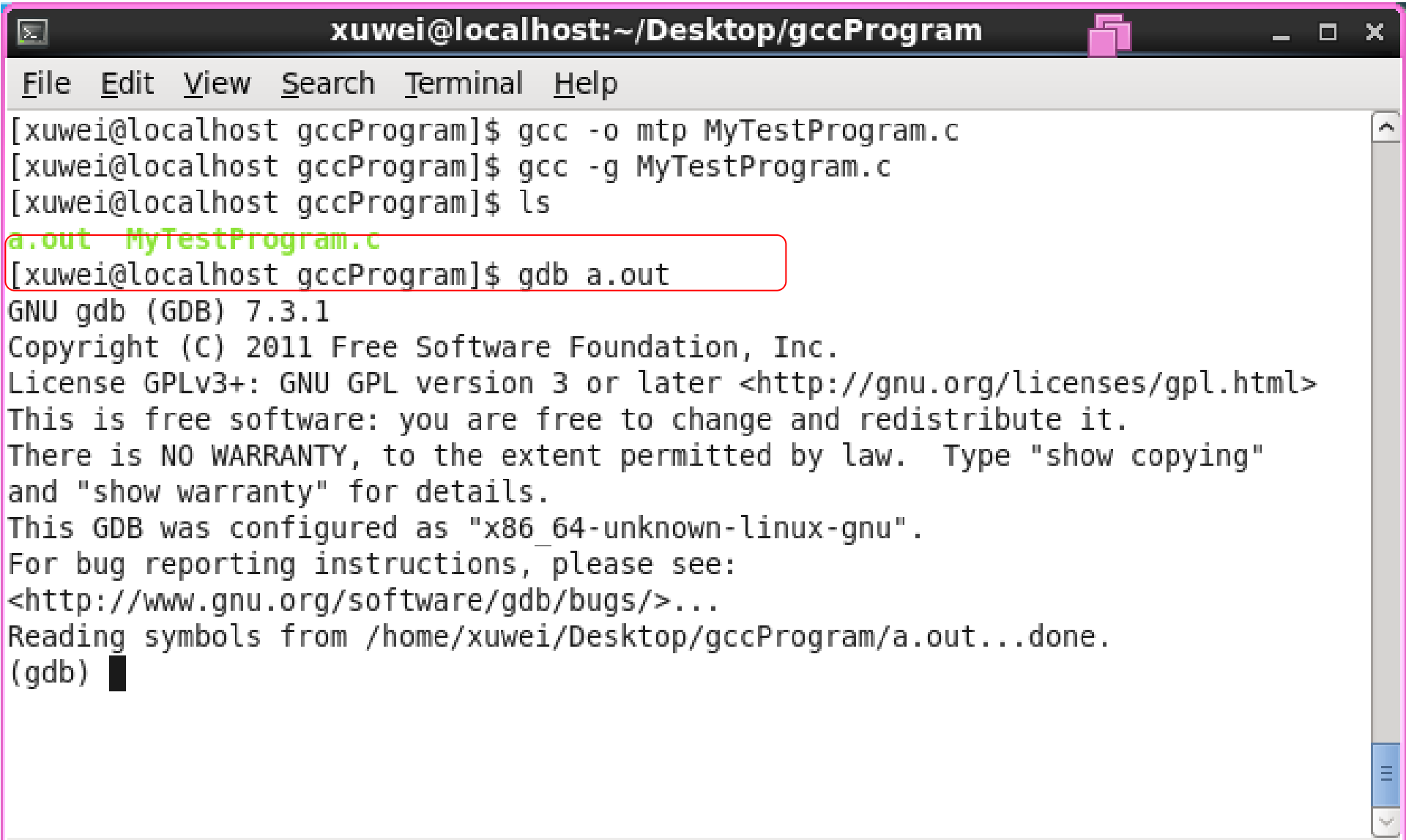
    file XXX (可执行文件)

  - **gdb XXX**

    (可执行文件)

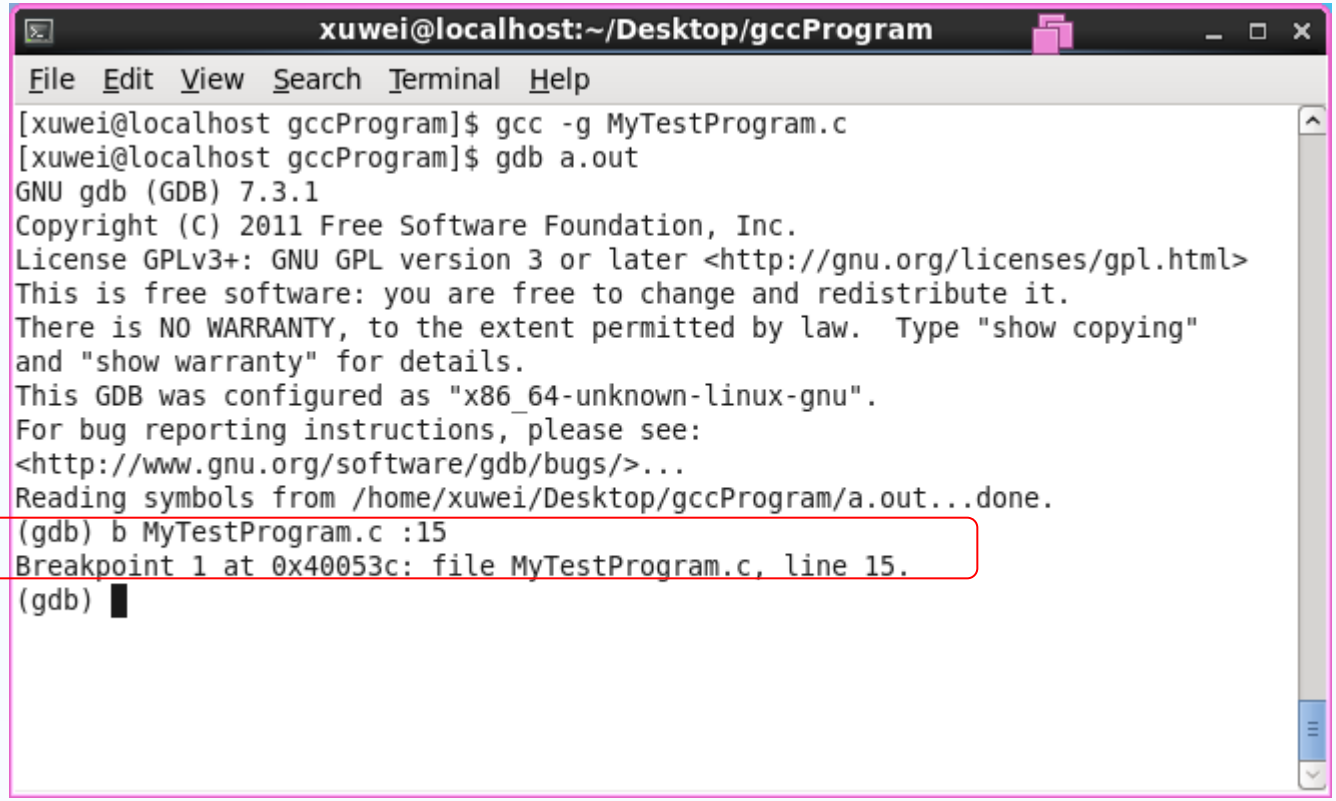


gdb入门

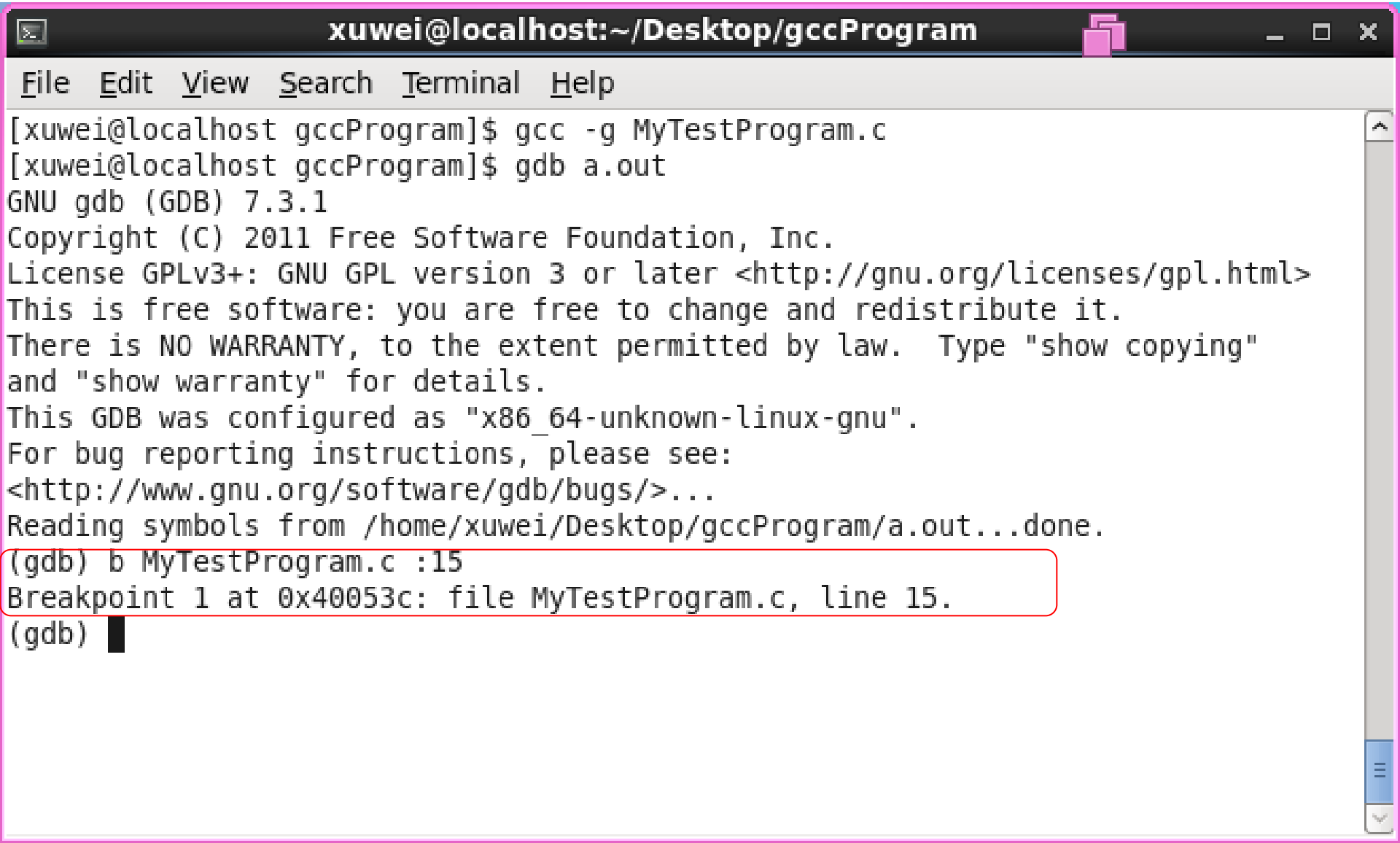

xuwei@localhost:~/Desktop/gccProgram
File Edit View Search Terminal Help
[xuwei@localhost gccProgram]$ gcc -o mtp MyTestProgram.c
[xuwei@localhost gccProgram]$ gcc -g MyTestProgram.c
[xuwei@localhost gccProgram]$ ls
a.out MyTestProgram.c
[xuwei@localhost gccProgram]$ gdb a.out
GNU gdb (GDB) 7.3.1
Copyright (C) 2011 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 "x86_64-unknown-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/xuwei/Desktop/gccProgram/a.out...done.
(gdb)

gdb

- 断点设置
  - break 文件名: 行号　　b 文件名: 行号



gdb入门


xuwei@localhost:~/Desktop/gccProgram
File Edit View Search Terminal Help
[xuwei@localhost gccProgram]$ gcc -g MyTestProgram.c
[xuwei@localhost gccProgram]$ gdb a.out
GNU gdb (GDB) 7.3.1
Copyright (C) 2011 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 "x86_64-unknown-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/xuwei/Desktop/gccProgram/a.out...done.
(gdb) b MyTestProgram.c :15
Breakpoint 1 at 0x40053c: file MyTestProgram.c, line 15.
(gdb)

gdb入门

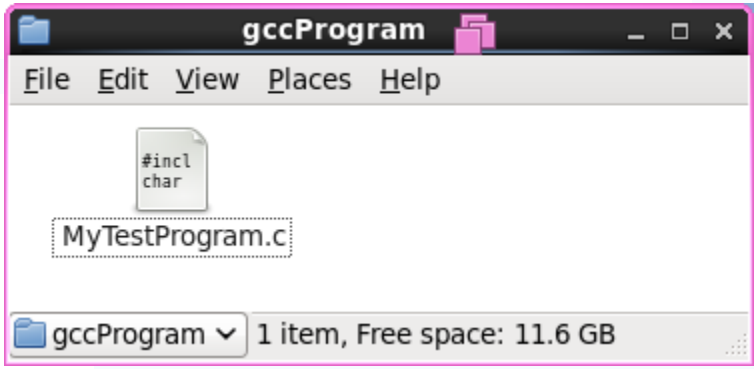
```
#include <stdio.h>
char week[7][10]={"Saturday", "Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday"};

int year[2]={365,366};
//year[0]表示非闰年的天数，year[1]表示闰年的天数。

int month[2][12]={31,28,31,30,31,30,31,31,30,31,30,31,
31,29,31,30,31,30,31,31,30,31,30,31};
//month[0]表示非闰年里每个月的天数，month[1]表示闰年里每个月的天数。

int type(int m){

    //判断第m年是否是闰年，是则返回1，否则返回0。
    if(m % 4 != 0 || (m % 100 == 0 && m % 400 != 0))        return  0; //不是闰年
    else return 1;   // 是闰年

}
```

Line:15

```
int  main(void)
{
    int days, dayofweek;
 //days 表示输入的天数，dayofweek表示星期几。
    int i = 0, j = 0;
    while (scanf("%d", &days) && days != -1) {
        dayofweek = days % 7;
        for(i = 2000; days >= year[type(i)]; i++)
            days -= year[type(i)];
        for(j = 0; days >= month[ type(i) ][ j ]; j++)
            days -= month[ type(i) ][ j ];
        printf("%d-%02d-%02d %s\n",
            i, j + 1, days + 1, week[dayofweek]);
    }
    return 0;
}
```

Line:25

Line:27



gdb入门

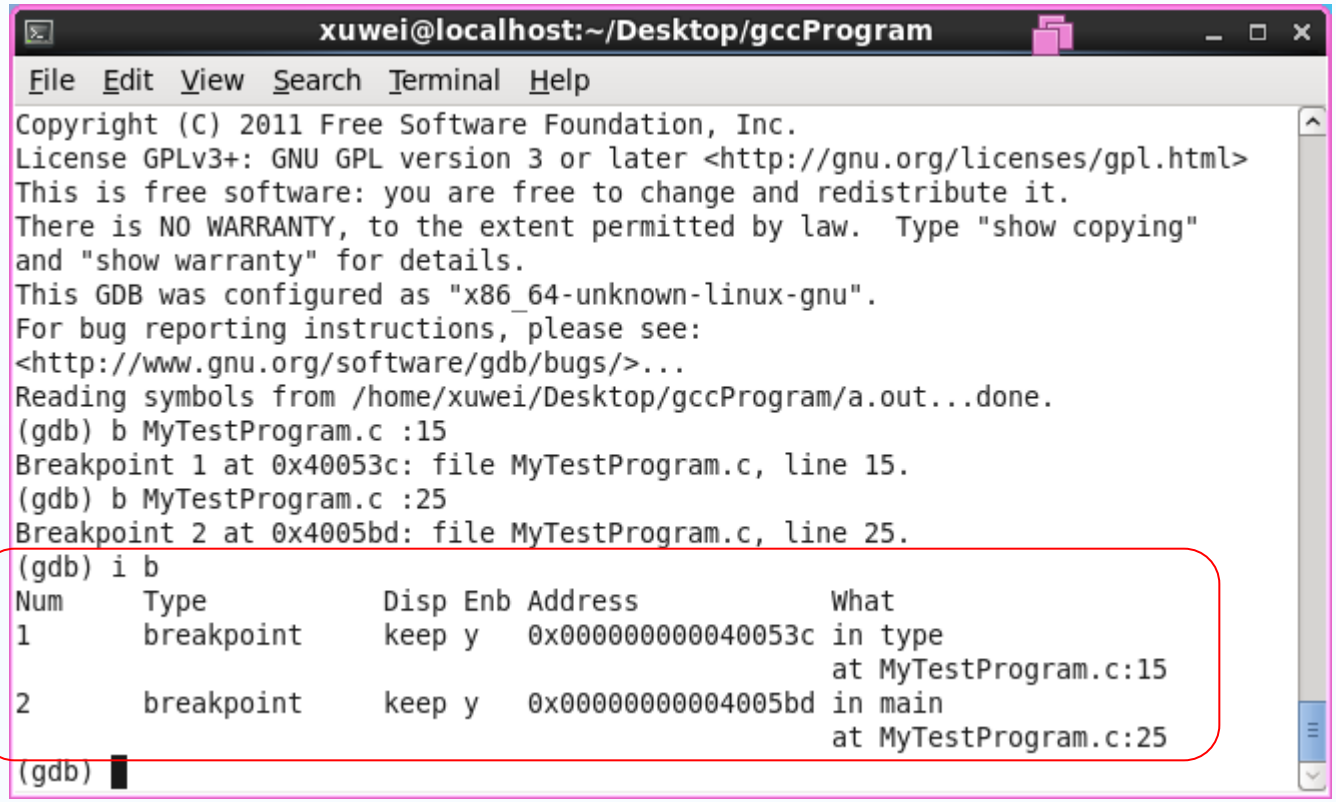
xuwei@localhost:~/Desktop/gccProgram

File Edit View Search Terminal Help

```
[xuwei@localhost gccProgram]$ gcc -g MyTestProgram.c
[xuwei@localhost gccProgram]$ gdb a.out
GNU gdb (GDB) 7.3.1
Copyright (C) 2011 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 "x86_64-unknown-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/xuwei/Desktop/gccProgram/a.out...done.
(gdb) b MyTestProgram.c :15
Breakpoint 1 at 0x40053c: file MyTestProgram.c, line 15.
(gdb) b MyTestProgram.c :25
Breakpoint 2 at 0x4005bd: file MyTestProgram.c, line 25.
(gdb)
```

gdb入门

- 断点查看
  - info b　　i b



gdb入门

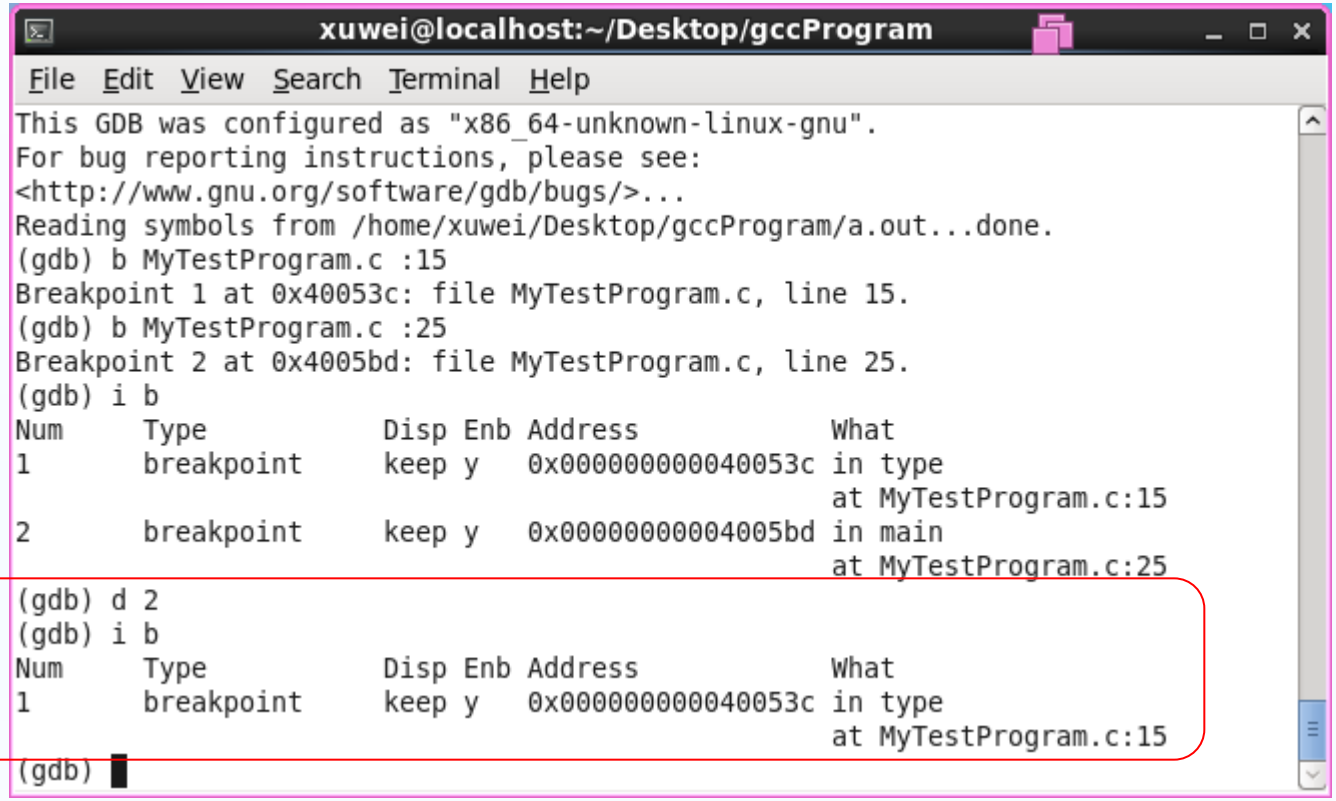
xuwei@localhost:~/Desktop/gccProgram

File Edit View Search Terminal Help

```
Copyright (C) 2011 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 "x86_64-unknown-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/xuwei/Desktop/gccProgram/a.out...done.
(gdb) b MyTestProgram.c :15
Breakpoint 1 at 0x40053c: file MyTestProgram.c, line 15.
(gdb) b MyTestProgram.c :25
Breakpoint 2 at 0x4005bd: file MyTestProgram.c, line 25.
(gdb) i b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x000000000040053c in type
                                                   at MyTestProgram.c:15
2       breakpoint     keep y   0x00000000004005bd in main
                                                   at MyTestProgram.c:25
(gdb)
```

gdb入门

- 断点删除
  - delete xxx(断点编号)　　d xxx(断点编号)



gdb入门

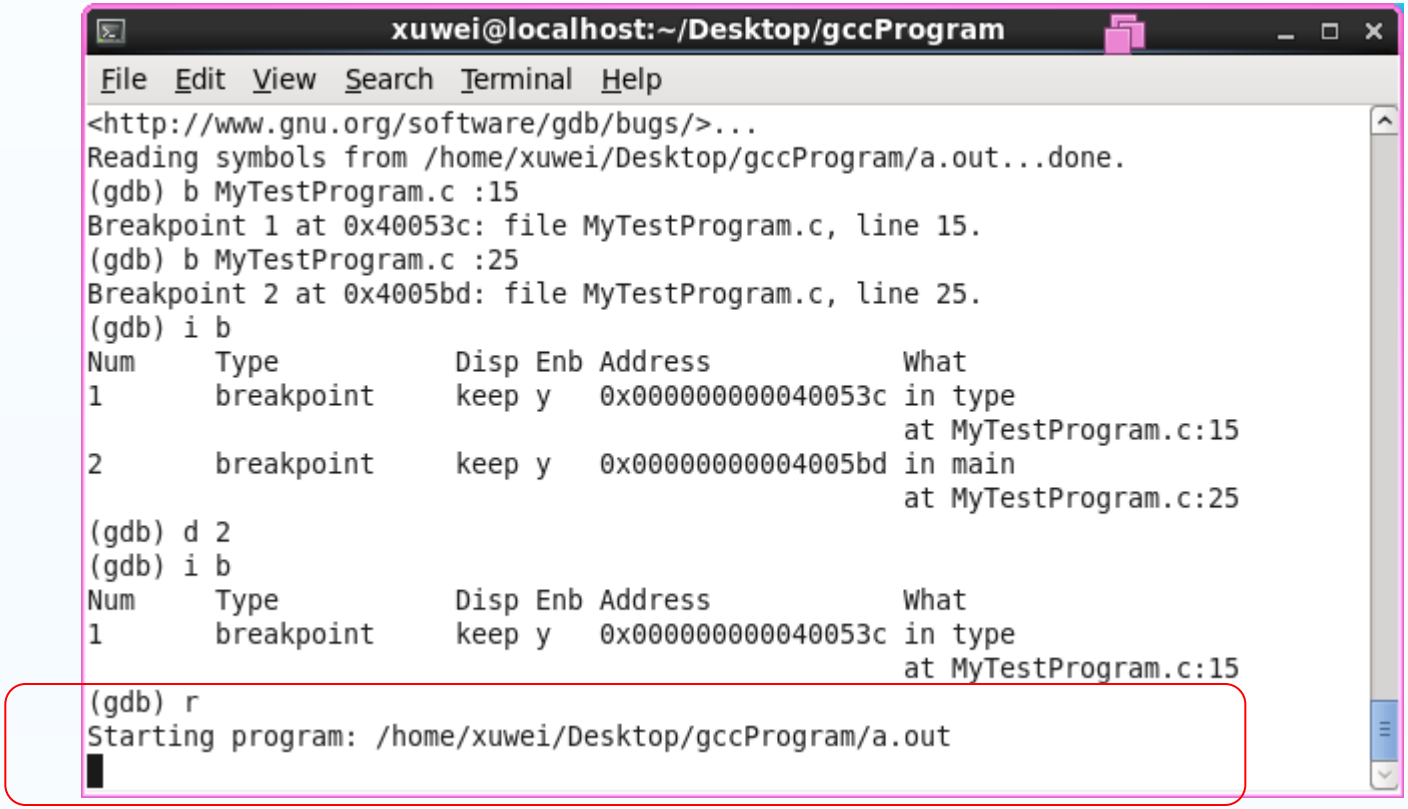
xuwei@localhost:~/Desktop/gccProgram

File Edit View Search Terminal Help

```
This GDB was configured as "x86_64-unknown-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/xuwei/Desktop/gccProgram/a.out...done.
(gdb) b MyTestProgram.c :15
Breakpoint 1 at 0x40053c: file MyTestProgram.c, line 15.
(gdb) b MyTestProgram.c :25
Breakpoint 2 at 0x4005bd: file MyTestProgram.c, line 25.
(gdb) i b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x000000000040053c in type
                                                   at MyTestProgram.c:15
2       breakpoint     keep y   0x00000000004005bd in main
                                                   at MyTestProgram.c:25
(gdb) d 2
(gdb) i b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x000000000040053c in type
                                                   at MyTestProgram.c:15
(gdb)
```

gdb入门

- 调试运行
  - run　　r



gdb入门

xuwei@localhost:~/Desktop/gccProgram

File Edit View Search Terminal Help

```
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/xuwei/Desktop/gccProgram/a.out...done.
(gdb) b MyTestProgram.c :15
Breakpoint 1 at 0x40053c: file MyTestProgram.c, line 15.
(gdb) b MyTestProgram.c :25
Breakpoint 2 at 0x4005bd: file MyTestProgram.c, line 25.
(gdb) i b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x000000000040053c in type
                                                   at MyTestProgram.c:15
2       breakpoint     keep y   0x00000000004005bd in main
                                                   at MyTestProgram.c:25
(gdb) d 2
(gdb) i b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x000000000040053c in type
                                                   at MyTestProgram.c:15
(gdb) r
Starting program: /home/xuwei/Desktop/gccProgram/a.out
```

gdb入门

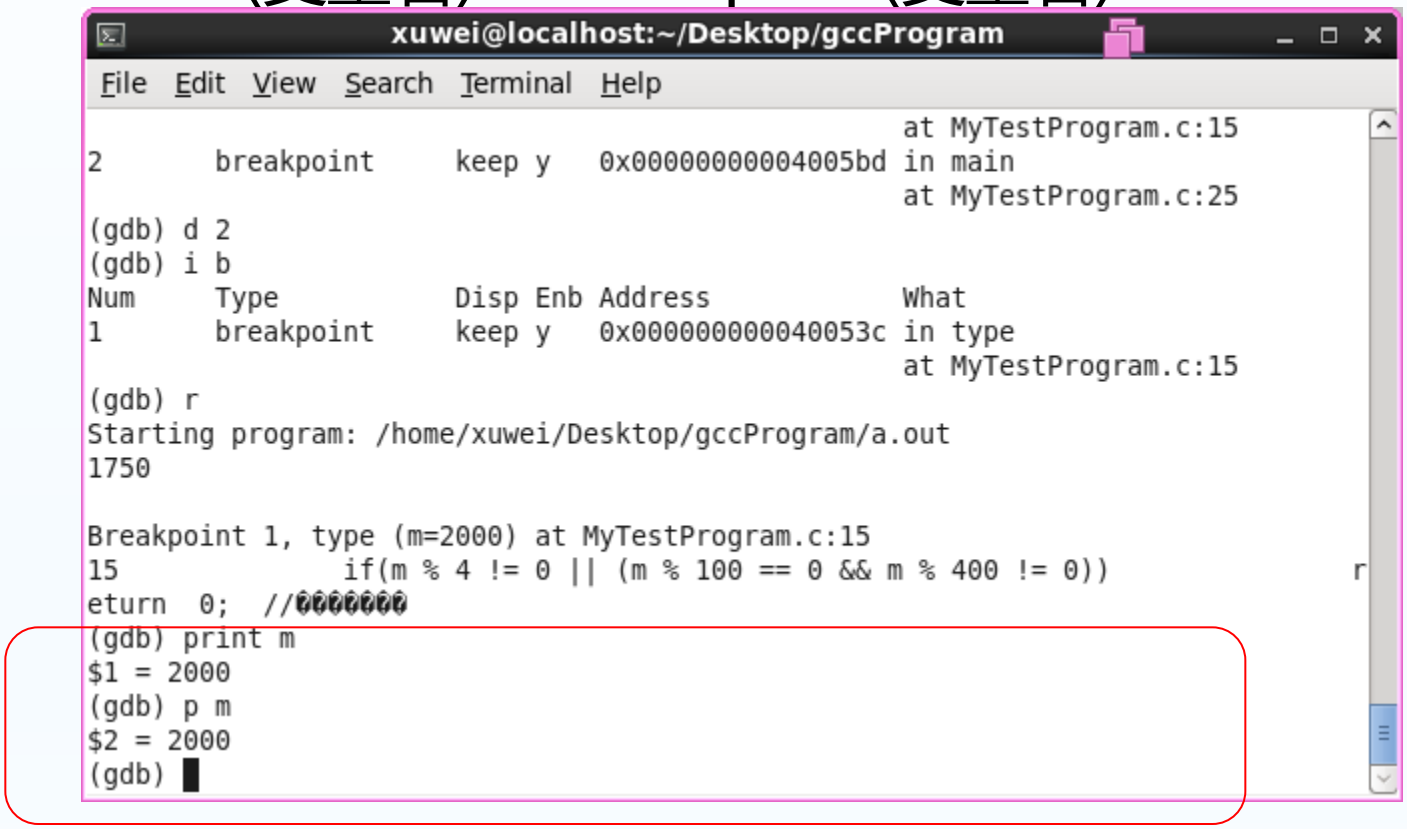
xuwei@localhost:~/Desktop/gccProgram

File Edit View Search Terminal Help

```
Breakpoint 2 at 0x4005bd: file MyTestProgram.c, line 25.
(gdb) i b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x000000000040053c in type
                                                   at MyTestProgram.c:15
2       breakpoint     keep y   0x00000000004005bd in main
                                                   at MyTestProgram.c:25
(gdb) d 2
(gdb) i b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x000000000040053c in type
                                                   at MyTestProgram.c:15
(gdb) r
Starting program: /home/xuwei/Desktop/gccProgram/a.out
1750

Breakpoint 1, type (m=2000) at MyTestProgram.c:15
15              if(m % 4 != 0 || (m % 100 == 0 && m % 400 != 0))
eturn  0;  //
(gdb)
```

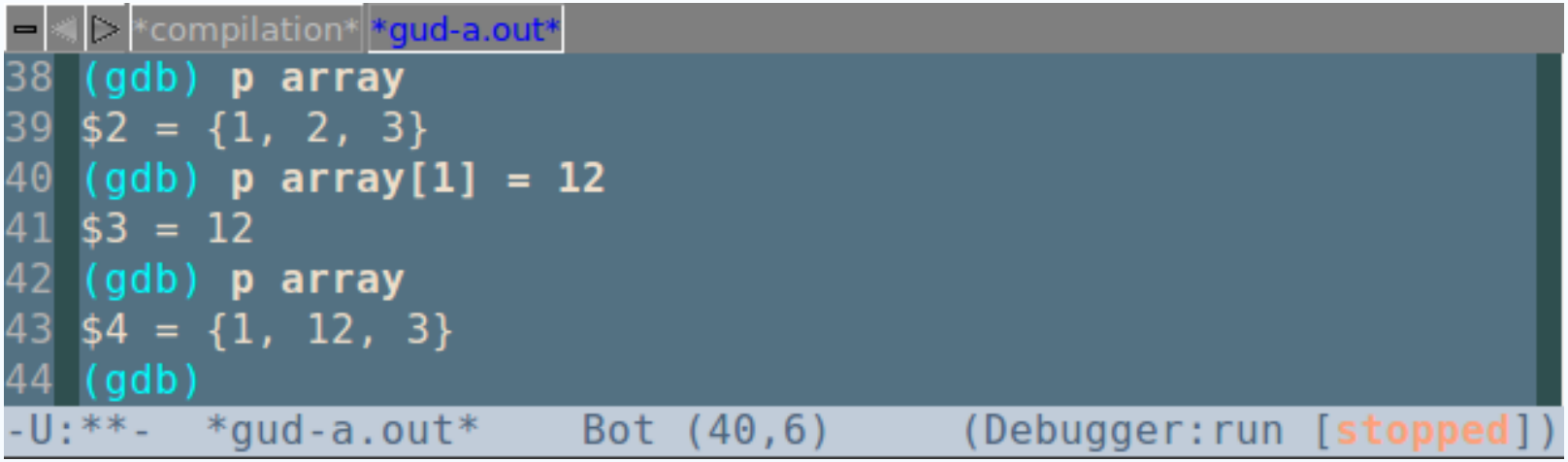
gdb入门

- 参数值查看
  - print xxx(变量名)　　p xxx(变量名)



gdb入门

- 修改参数值
  - print xxx(变量名) = XXX
  - p xxx(变量名) = XXX



gdb入门

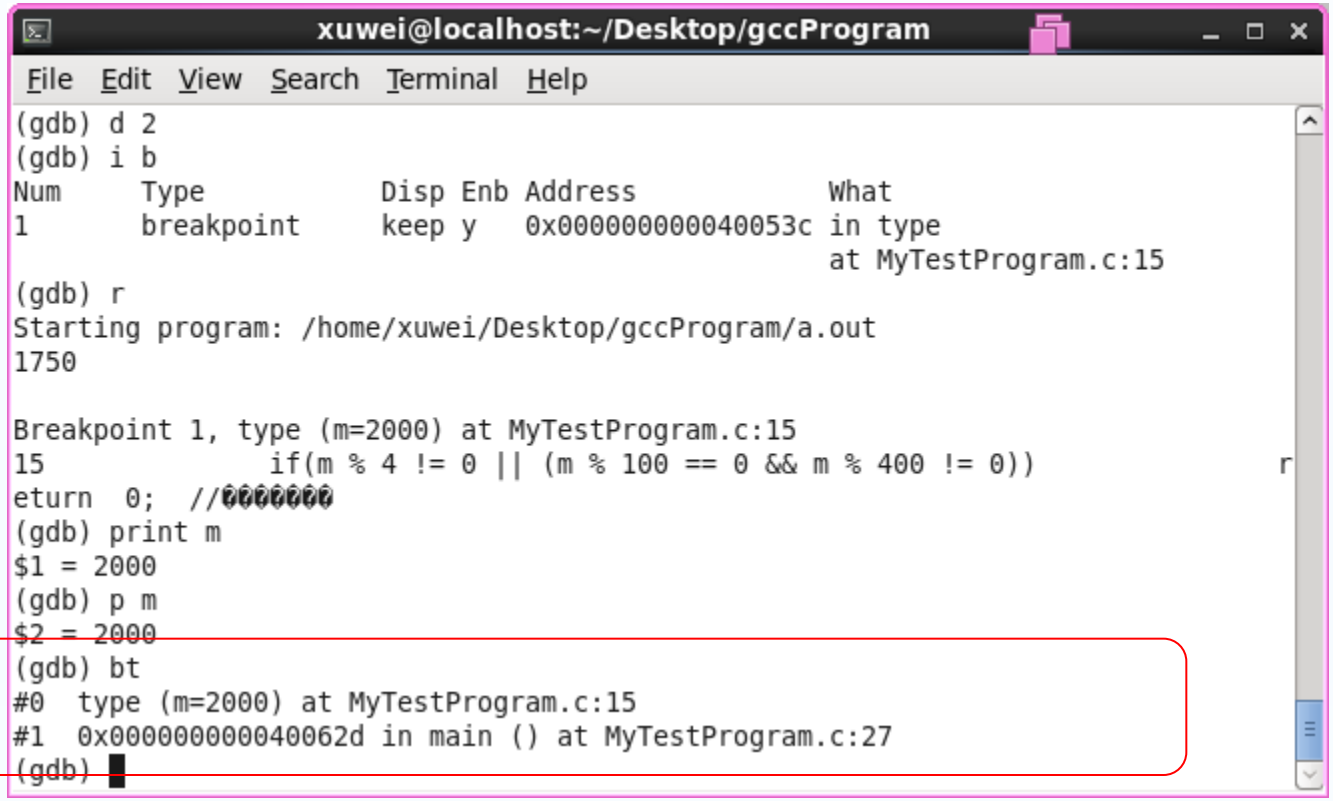
xuwei@localhost:~/Desktop/gccProgram

File Edit View Search Terminal Help

```
                                                   at MyTestProgram.c:15
2       breakpoint     keep y   0x0000000004005bd in main
                                                   at MyTestProgram.c:25

(gdb) d 2
(gdb) i b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x000000000040053c in type
                                                   at MyTestProgram.c:15

(gdb) r
Starting program: /home/xuwei/Desktop/gccProgram/a.out
1750

Breakpoint 1, type (m=2000) at MyTestProgram.c:15
15              if(m % 4 != 0 || (m % 100 == 0 && m % 400 != 0))                r
eturn  0;  //
(gdb) print m
$1 = 2000
(gdb) p m
$2 = 2000
(gdb)
```

gdb入门

- 函数调用堆栈
  - bt



gdb入门

xuwei@localhost:~/Desktop/gccProgram

File Edit View Search Terminal Help

```
(gdb) d 2
(gdb) i b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x000000000040053c in type
                                                   at MyTestProgram.c:15
(gdb) r
Starting program: /home/xuwei/Desktop/gccProgram/a.out
1750

Breakpoint 1, type (m=2000) at MyTestProgram.c:15
15              if(m % 4 != 0 || (m % 100 == 0 && m % 400 != 0))                r
eturn  0;  //
(gdb) print m
$1 = 2000
(gdb) p m
$2 = 2000
(gdb) bt
#0  type (m=2000) at MyTestProgram.c:15
#1  0x000000000040062d in main () at MyTestProgram.c:27
(gdb)
```

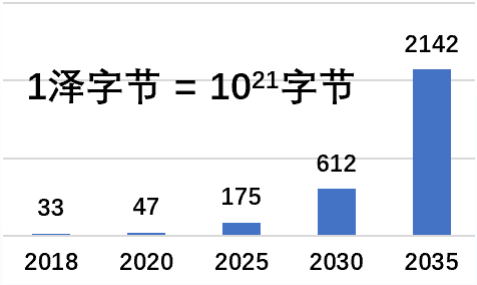
gdb入门

- 单步执行
  - next　　n
  - step　　s
- 继续执行到下一个断点
  - continuec
- 查看文件代码
  - list　　l
- 退出gdb
  - quit　　q

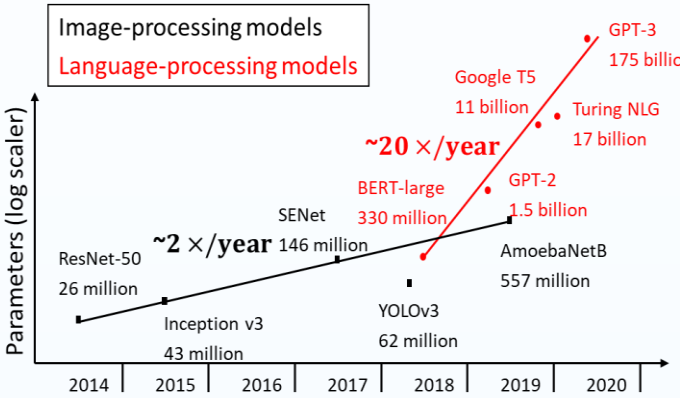
课程概述

- 程序设计回顾
- 常用编译工具使用
- 程序如何运行起来?
- 流程图与常用UML图

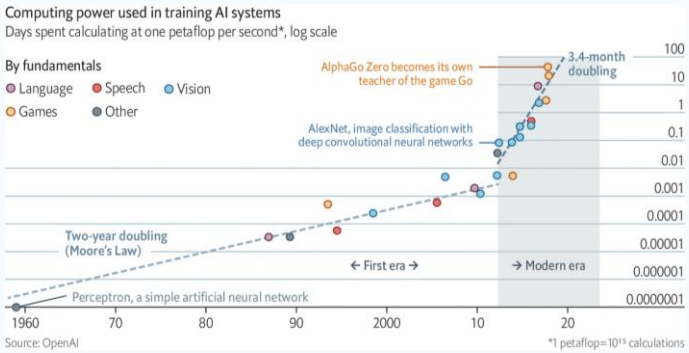
软件必须要懂硬件



数字经济、数字孪生助推全社会数据的爆炸式增长

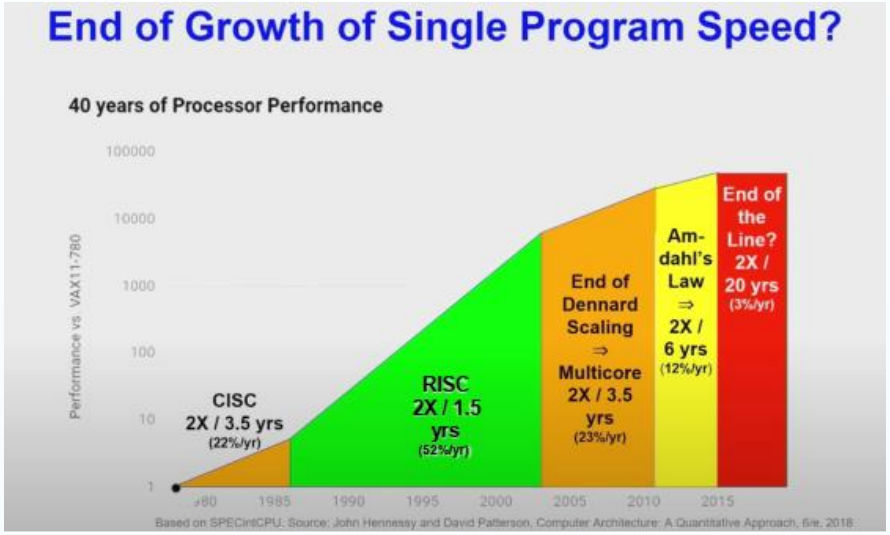


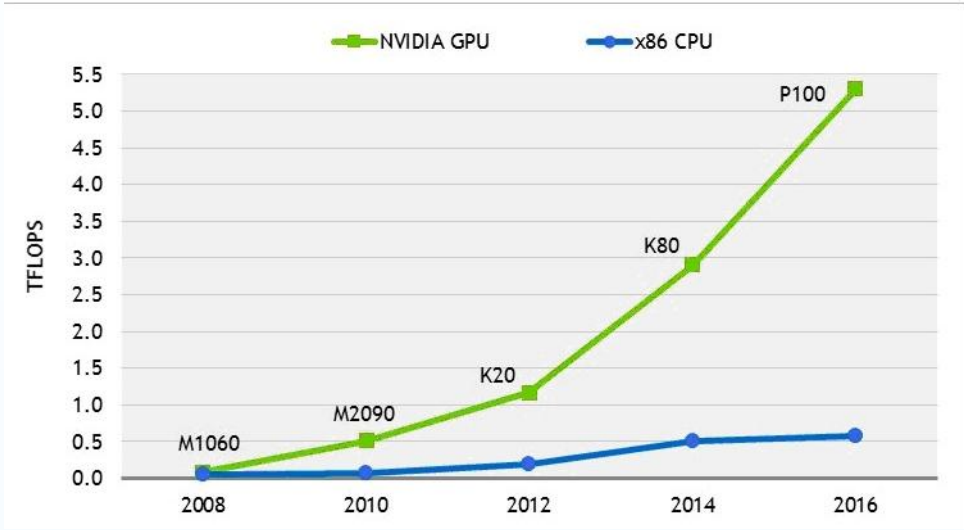
新版摩尔定律：
宇宙中的智能数量每18个月就会翻一倍



过去十年，AI训练算力需求增长了30万倍，3.4个月翻一倍

体系结构的黄金时代





Turing Lecture, Hennessy, Patterson; June 2018 / CACM Feb 2019

https://www.rtinsights.com/gpus-the-key-to-cognitive-computing/

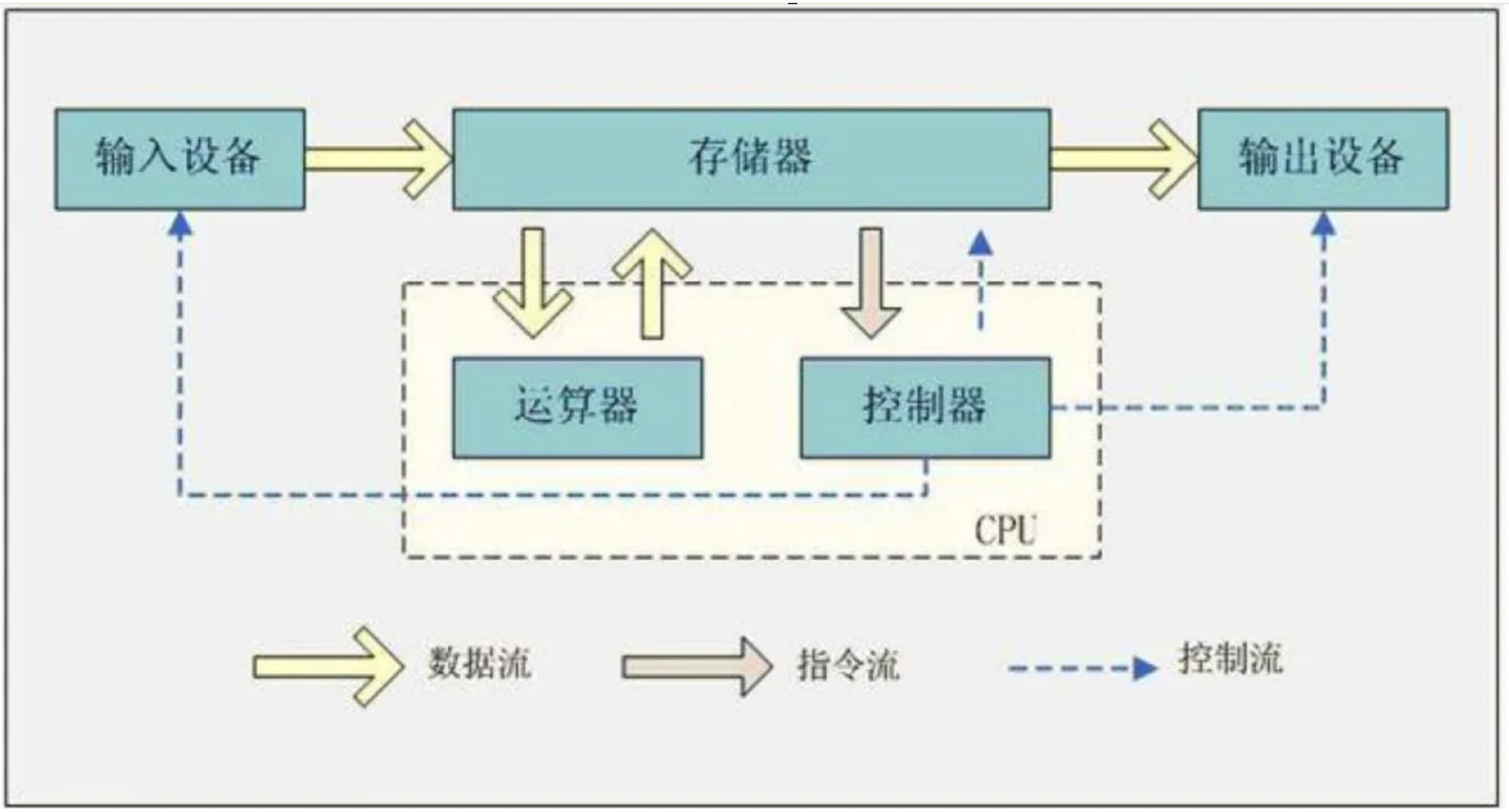
- 领域专用芯片的使用助推了人工智能的发展

体系结构黄金时代的机会

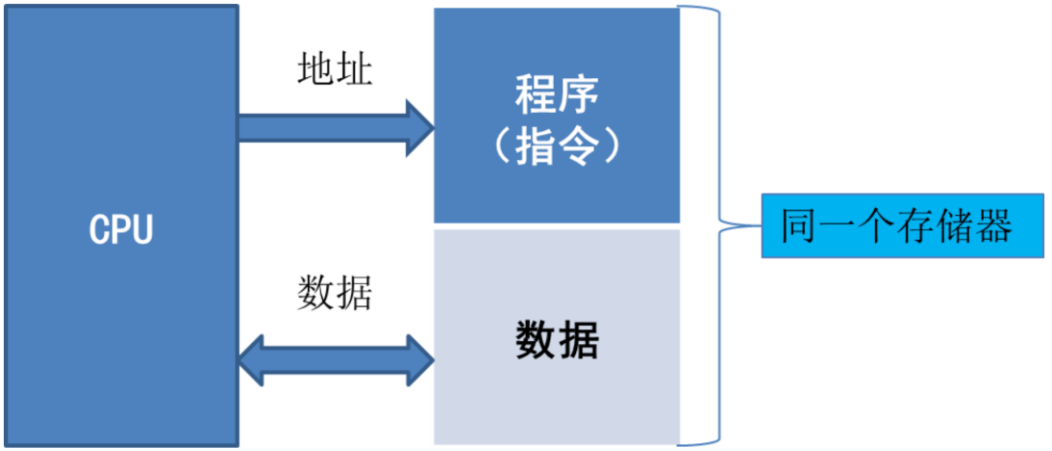
- 软件为中心
  - 现代编程语言很多是script语言-> 需要解析、动态类型、代码复用
  - 编程高效但是执行不见得高效：Python vs. C
- 硬件为中心
  - 唯一可行的路径是领域专用架构
  - 一颗芯片只做一部分事情，但是性能很好
- 软硬协同的设计思路
  - 专用领域语言+专用领域架构：CUDA + GPU
  - 可以最大化挖掘硬件的并行度

程序是如何运行起来的

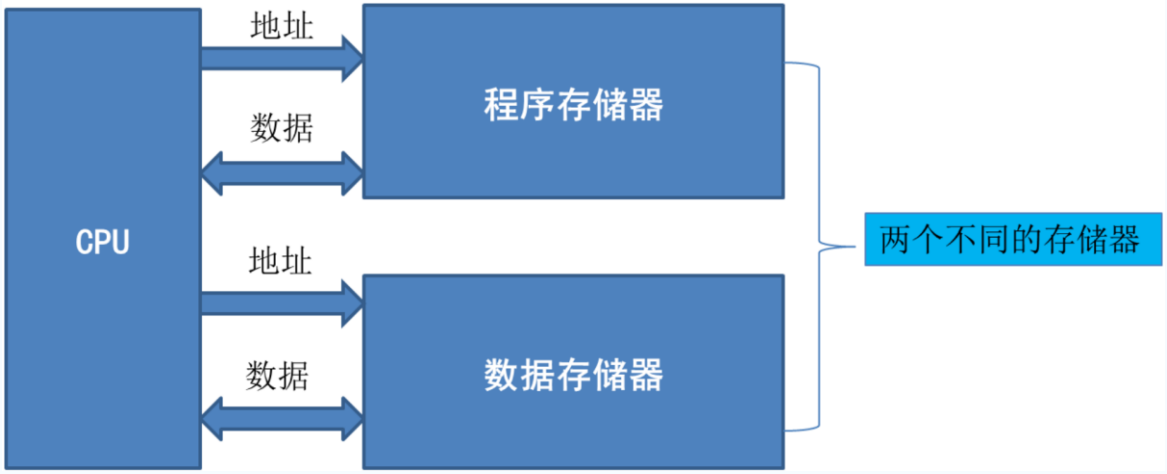
- 计算机



程序是如何运行起来的



冯诺依曼体系结构



哈佛体系结构

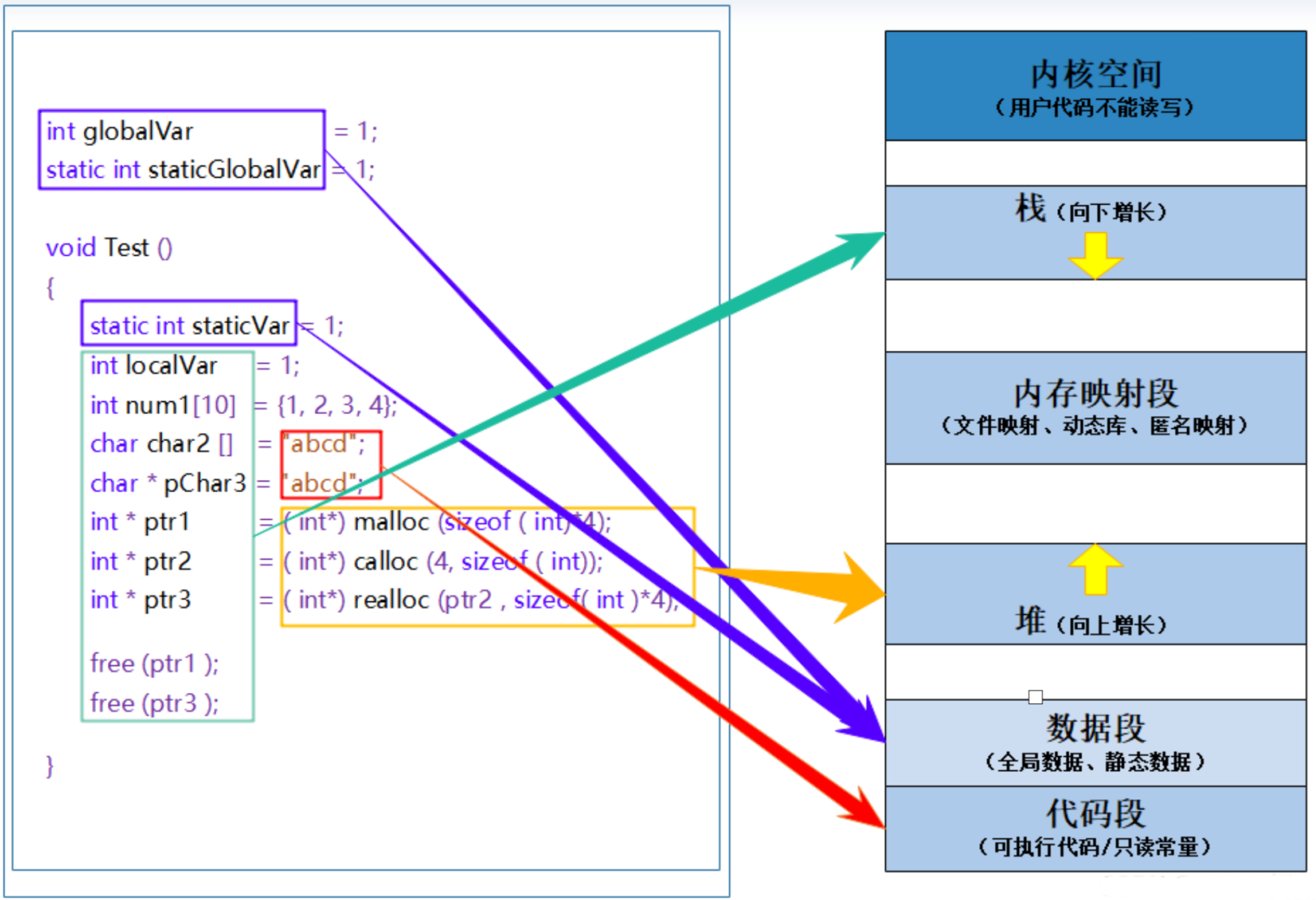
程序是如何运行起来的

- 内存分布

  32位与64位

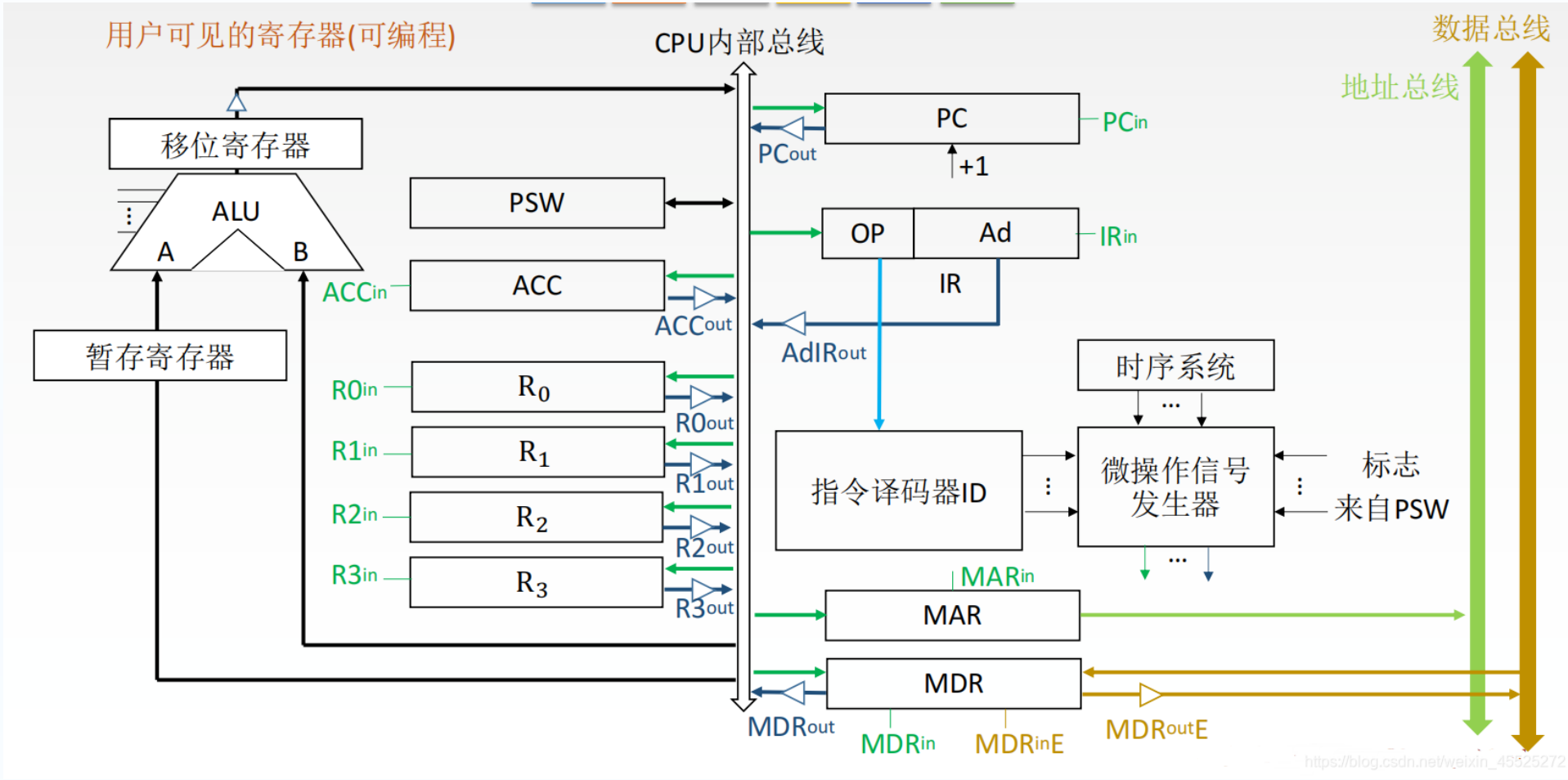
程序是如何运行起来的

- 内存



程序是如何运行起来的

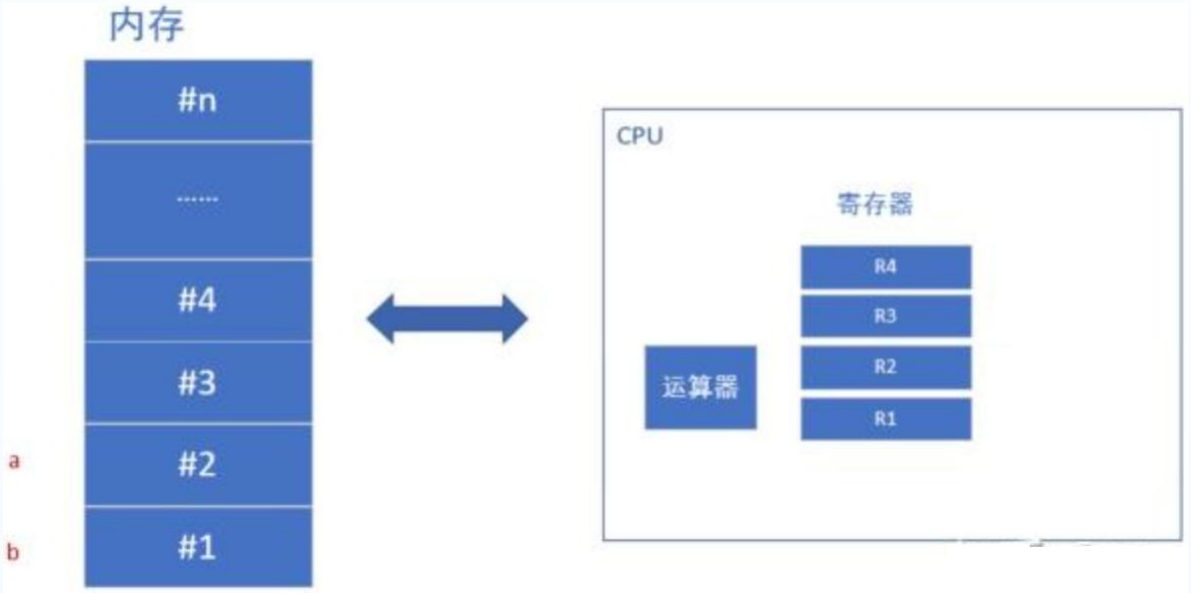
- CPU结构



程序是如何运行起来的

- CPU和内存的交互

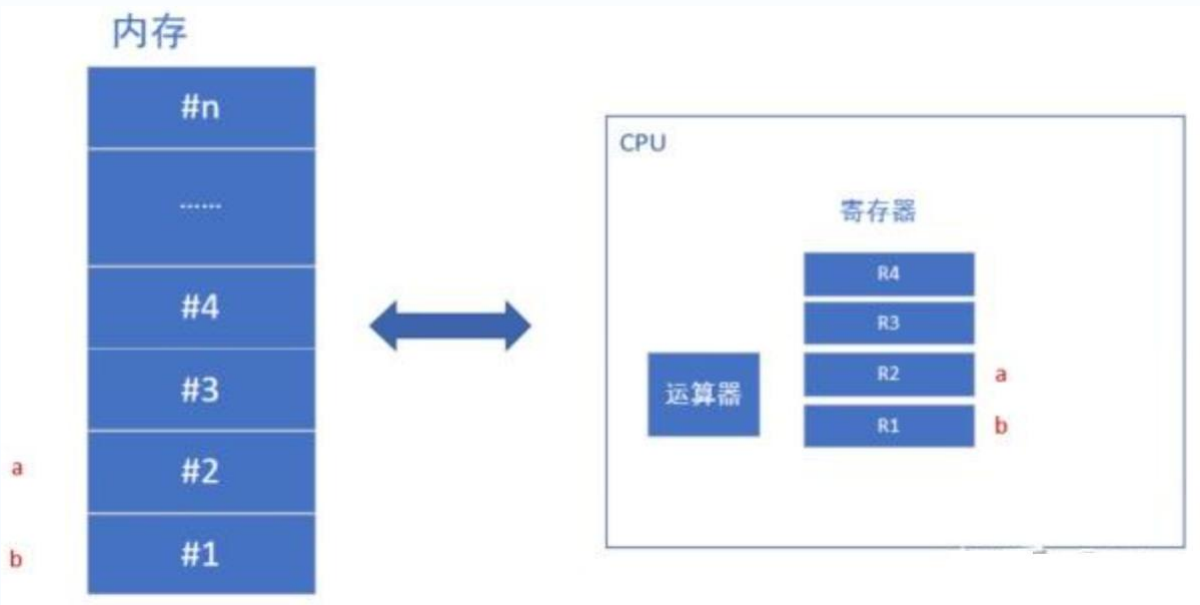
```
#include <stdio.h>
int main ()
{
   int a = 5, b = 3, sum;
   sum = a + b;
   printf("%d\n", sum);
   return 0;
}
```



程序是如何运行起来的

- CPU和内存的交互

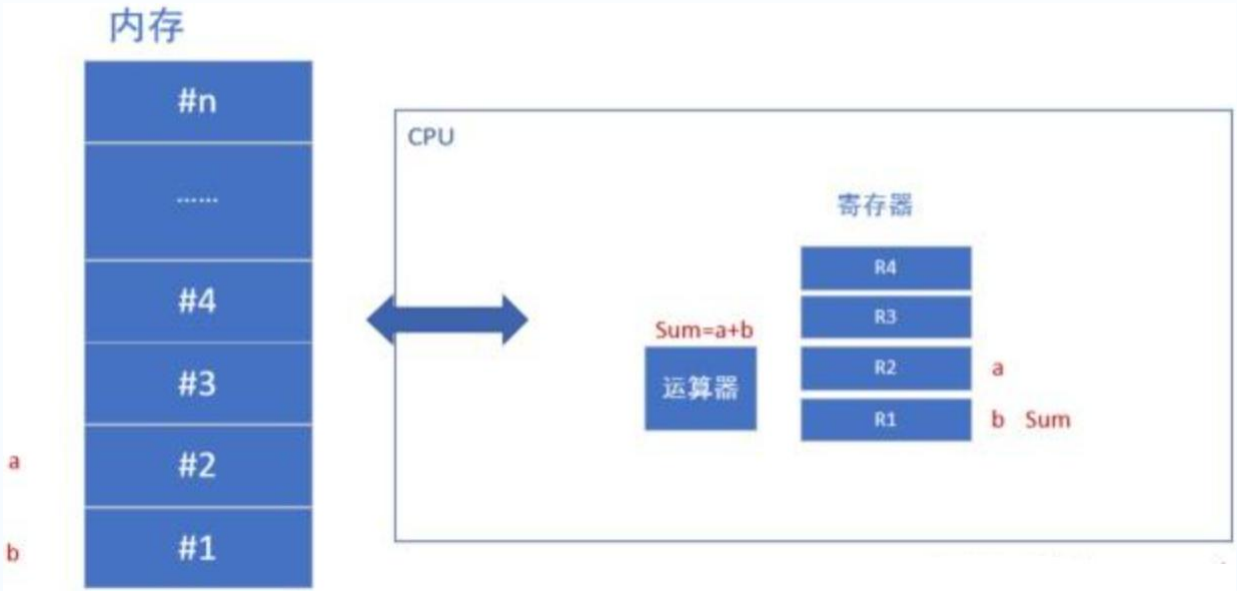
```
#include <stdio.h>
int main ()
{
   int a = 5, b = 3, sum;
   sum = a + b;
   printf("%d\n", sum);
   return 0;
}
```



程序是如何运行起来的

- CPU和内存的交互

```
#include <stdio.h>
int main ()
{
   int a = 5, b = 3, sum;
   sum = a + b;
   printf("%d\n", sum);
   return 0;
}
```



课程概述

- 程序设计回顾
- 常用编译工具使用
- 程序如何运行起来?
- 流程图与常用UML图
  - 流程图
  - 活动图

流程图

- 起源：工业工程与流程图表（1920s）
  - 最初是为了分析建筑工人的动作和工序，试图找到最有效的施工方法，减少不必要的动作
  - 这被认为是流程图的雏形
- 标准化：ASME与符号规范的建立（1940s）
  - 美国机械工程师学会在20世纪40年代开始推动流程图的标准化
  - 流程图已成为工业生产、商业管理和后勤领域中描述工作流的通用工具

流程图

- 引入计算机领域：冯·诺依曼与程序框图（1940s-1950s）
  - 流程图因其直观、清晰的特性，被引入计算机编程领域
  - 流程图是程序员最主要的设计和交流工具

- 黄金时代与标准化：ISO与ANSI（1960s-1970s）
  - 1970年，美国国家标准协会采用了基于之前工作的标准，制定了 **ANSI X3.5** 标准《信息系统的流程图符号》
  - 1970年，ISO也采纳了ANSI标准，使得流程图符号在全球范围内得到统一

流程图

- 现代演变：从流程图到UML活动图（1990s至今）
  - 1997年，统一建模语言发布。它将传统流程图的思想吸收并扩展，形成了 UML活动图（支持并发等特性）

流程图

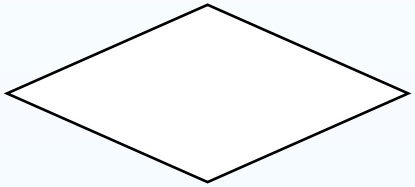
- 基本符号

开始/
结束

数据输入
输出

处理

准备或
预处理



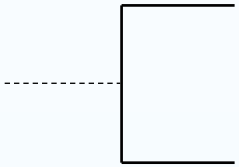
条件判断

文件或
文档

流程图

- 基本符号

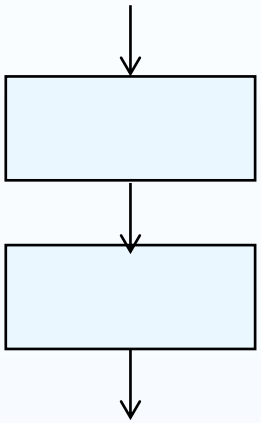
流线



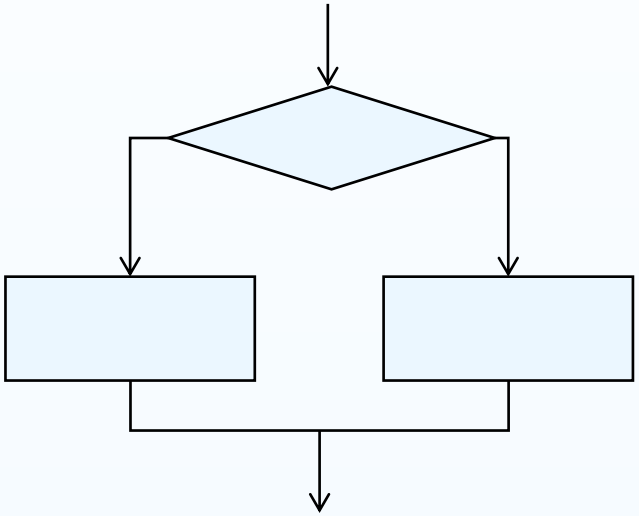
注解或注释

流程图

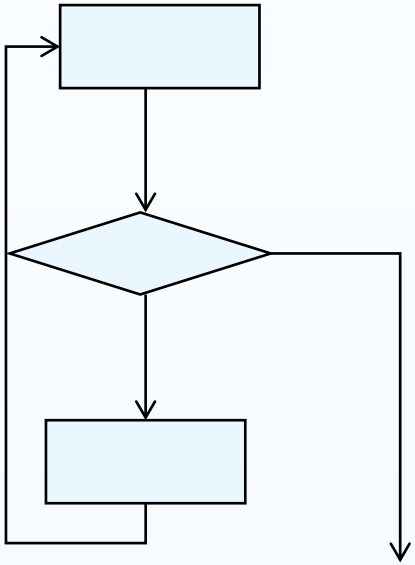
- 基本结构

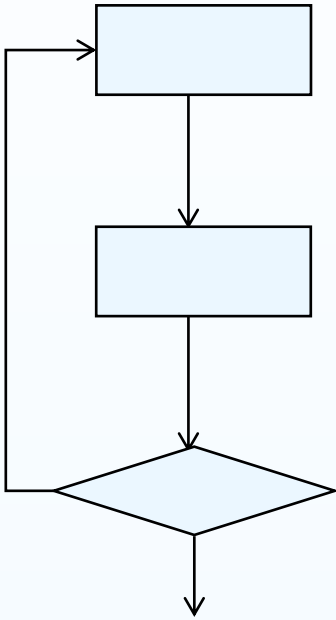


顺序



分支





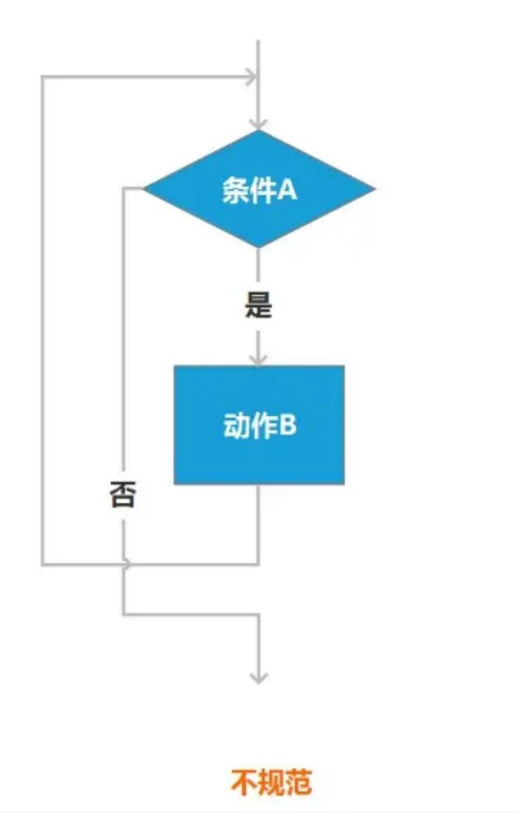
循环

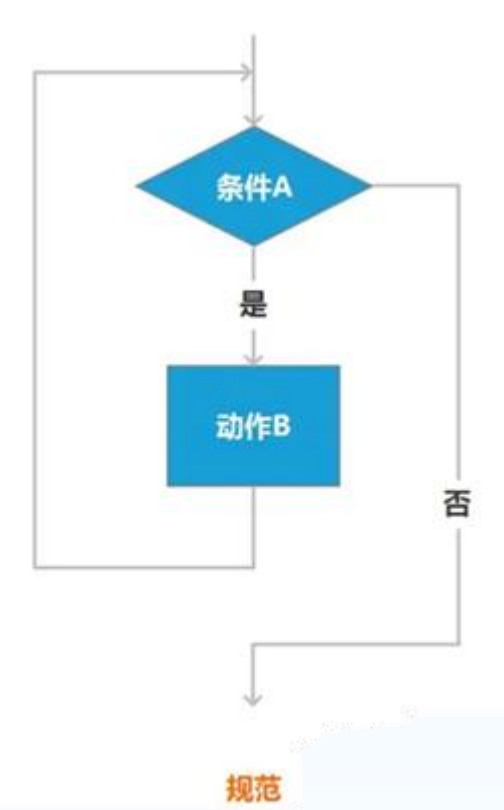
流程图

- 注意：
  - 请使用标准的框图符号，而不是自定义符号
  - 至少包括开始、结束和一个处理
  - 整体绘制方向是从上到下、从左往右
  - 除了判断框外，大多数框图只有一个流入和一个流出
  - 开始框只有一个流出线，终止框只有一个流入线
  - 判断框输出线需要标注判断条件
  - 线与线避免交叉
  - 语言尽可能精炼

流程图

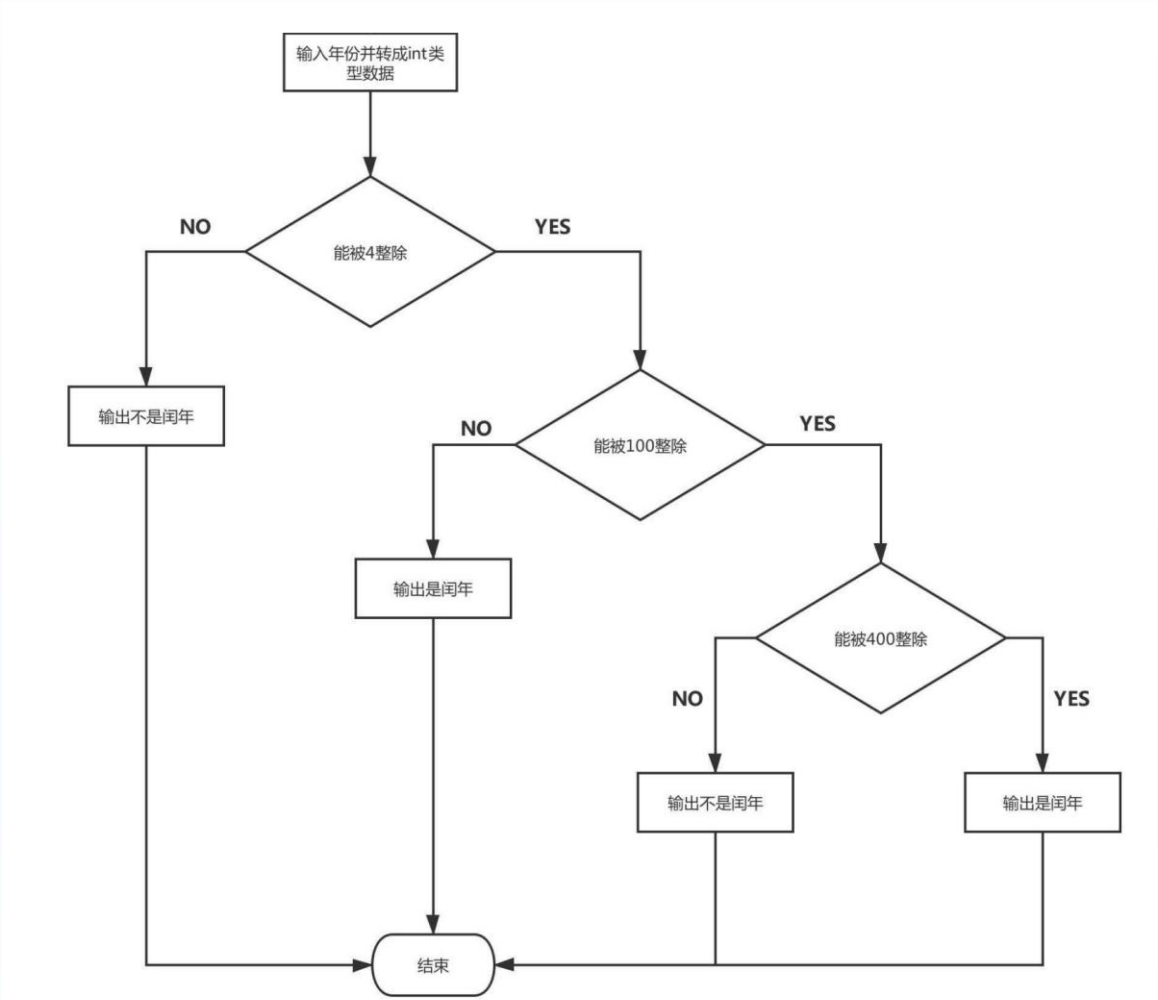
- 常见不规范：连接线交叉





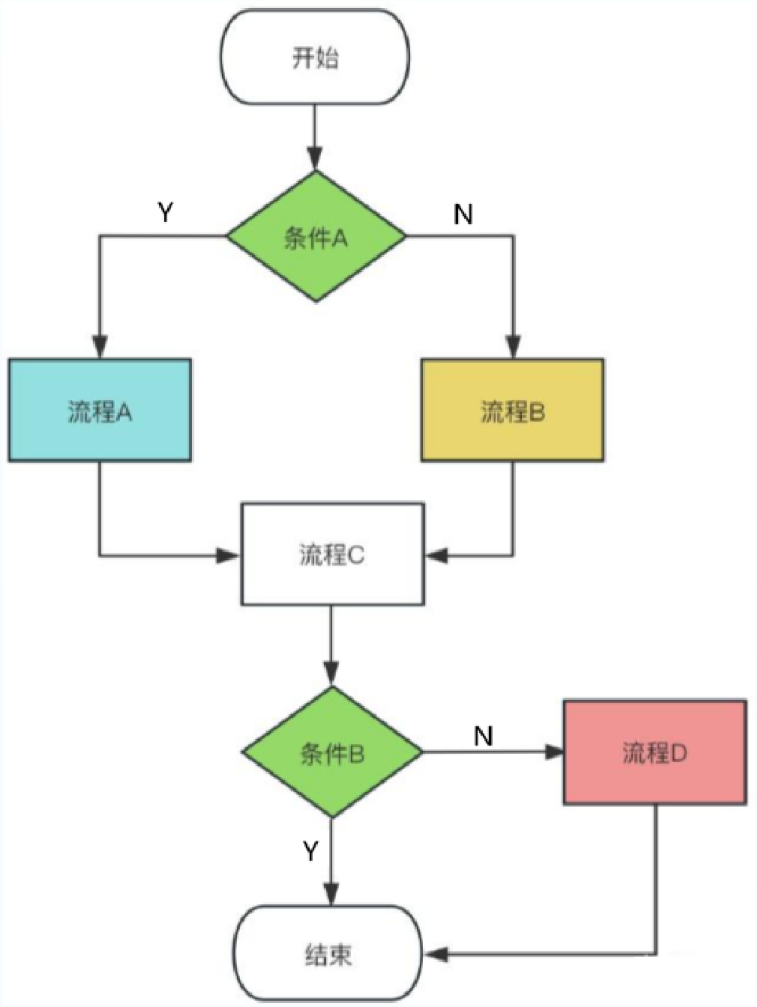
流程图

- 常见不规范：符号不规范



流程图

- 示例

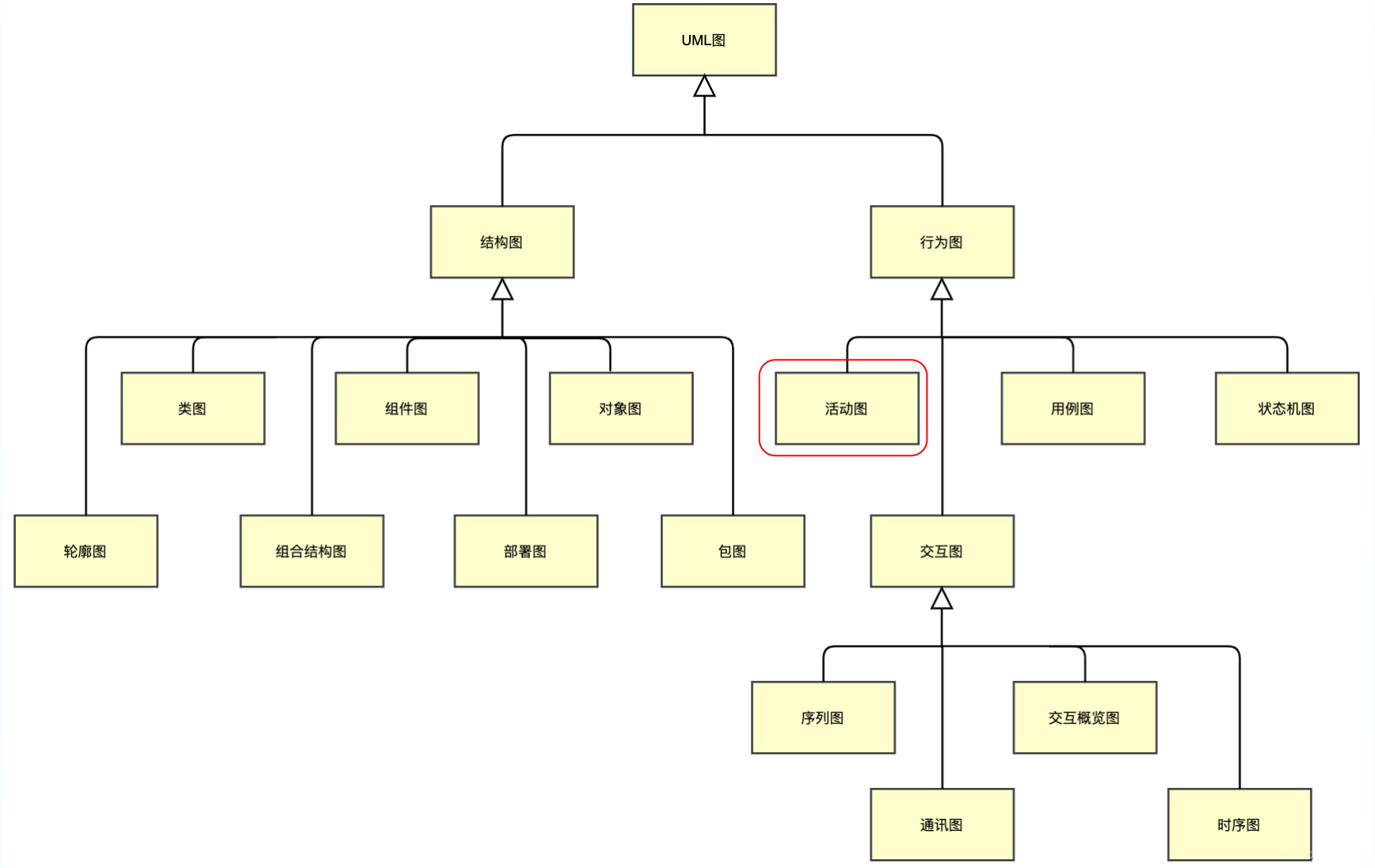


UML

- UML
  - Unified Model Language
  - 中文：统一建模语言
  - 是由一整套图表组成的标准化建模语言

- 为什么要UML
  - 对整个软件设计有更好的可读性，可理解性，从而降低开发风险
  - 方便各个开发人员之间的交流

UML


UML图
结构图
行为图
类图
组件图
对象图
活动图
用例图
状态机图
轮廓图
组合结构图
部署图
包图
交互图
序列图
交互概览图
通讯图
时序图

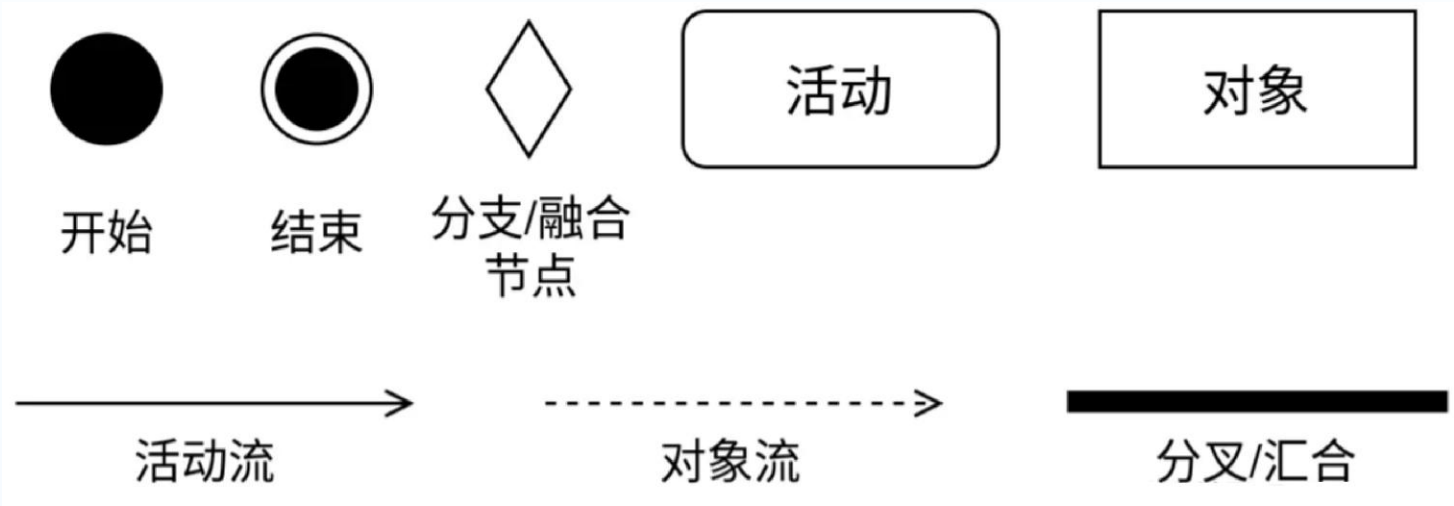
活动图

- UML活动图以图形化的方式展示**控制流**和**对象流**
- 强调从一个活动到另一个活动的顺序、分支和并发
- 常用于描述内部的工作流程、业务过程或操作的详细逻辑

- 与流程图的区别
  - 并发建模
    - 通过分叉和汇合节点直接支持并发流程，传统流程图通常只能描述单一线程
  - 责任划分（引入泳道）
  - 面向对象（与用例图、类图衔接）

活动图

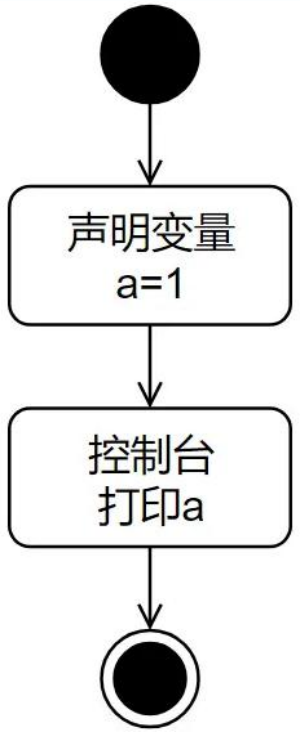
- 基本符号



活动图

- 简单顺序结构

```
#include <stdio.h>

void main() {
    int a = 1;
    printf("a = %d\n", a);
}
```

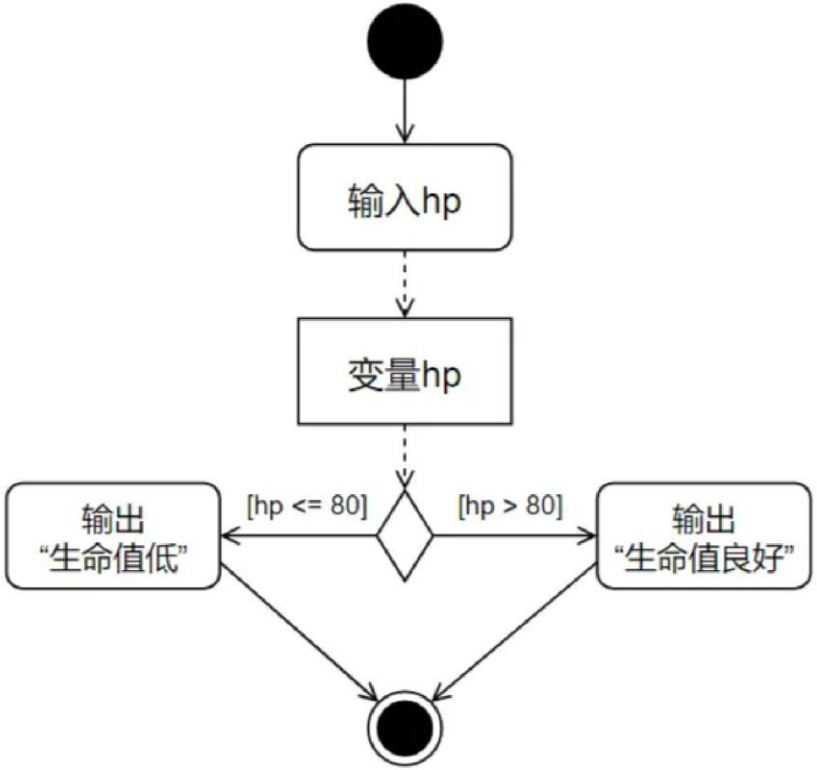


与流程图类似，每个步骤表示为一个活动
活动之间使用**单向实线箭头连线**表示它们的执行顺序

活动图

- 分支结构

```
#include <stdio.h>

void main() {
    int hp;
    scanf("%d", &hp);
    if(hp > 80)
        printf("生命值良好");
    else
        printf("生命值低");
}
```

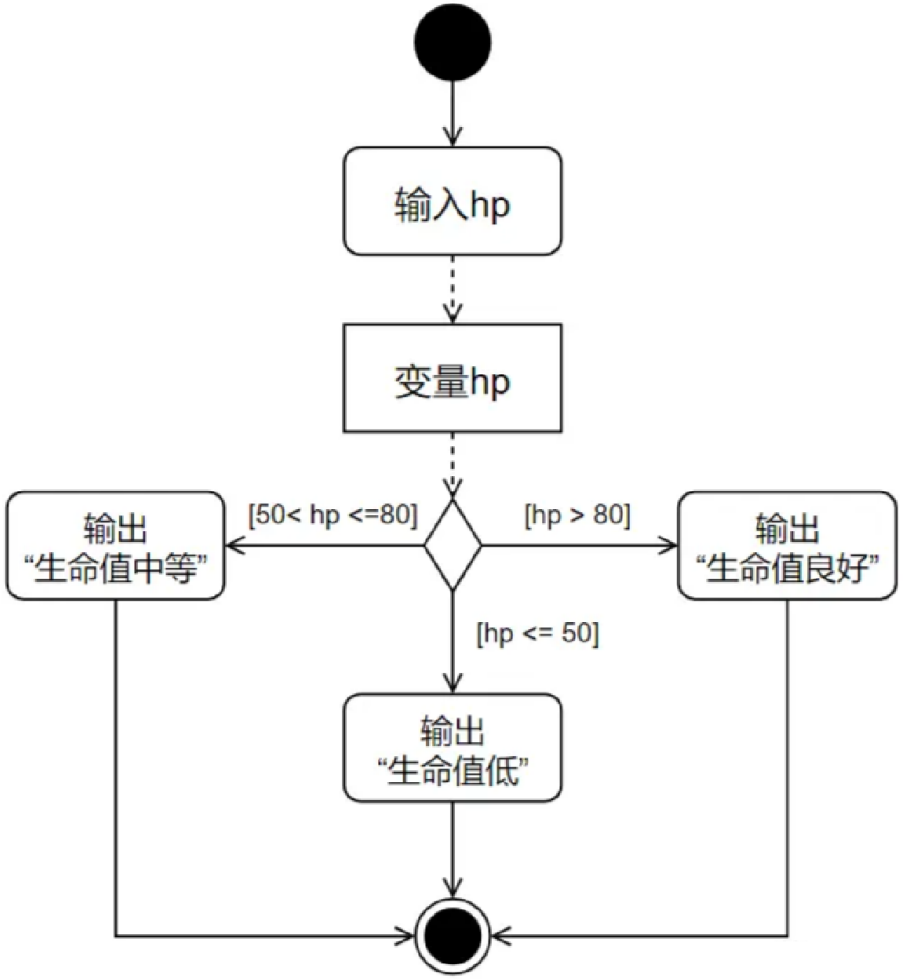


分支条件标记在从菱形延伸出去的线上，使用方括号[]包围条件

活动图

- 分支结构

```
#include <stdio.h>

void main() {
    int hp;
    scanf("%d", &hp);
    if(hp > 80)
        printf("生命值良好");
    else if(hp > 50)
        printf("生命值中等");
    else
        printf("生命值低");
}
```

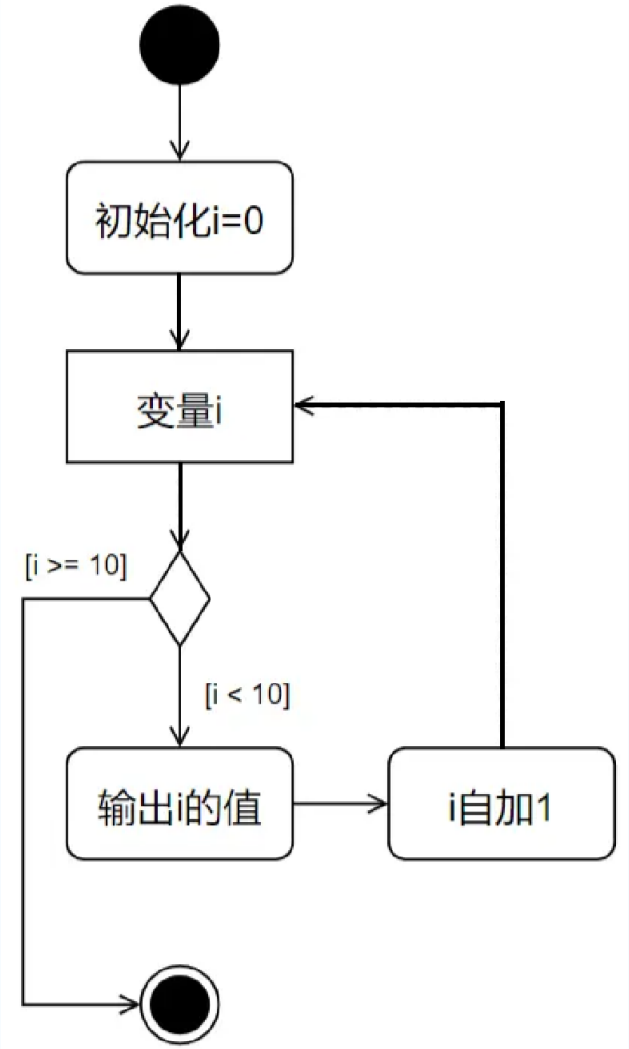


分支不仅仅能够表示二元条件，也可以表示多元条件判断

活动图

- 循环结构

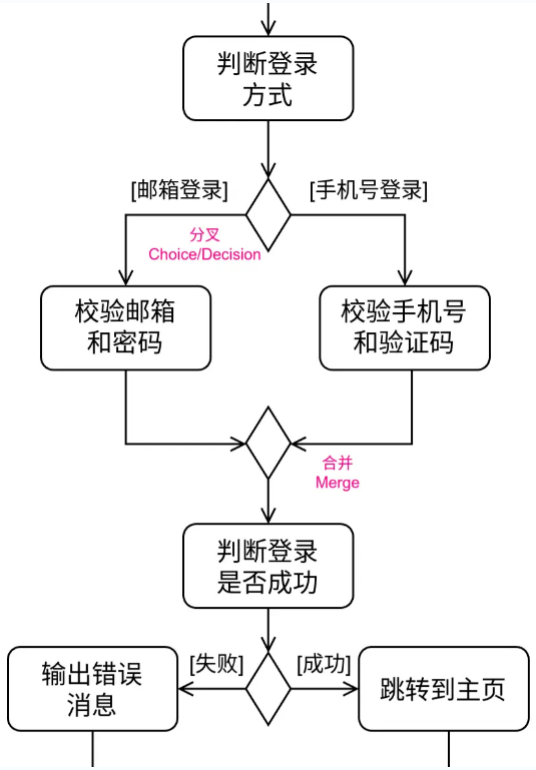
```
...
for(int i = 0; i < 10; i ++)
    printf(" %d\n", i);
...
```



活动图

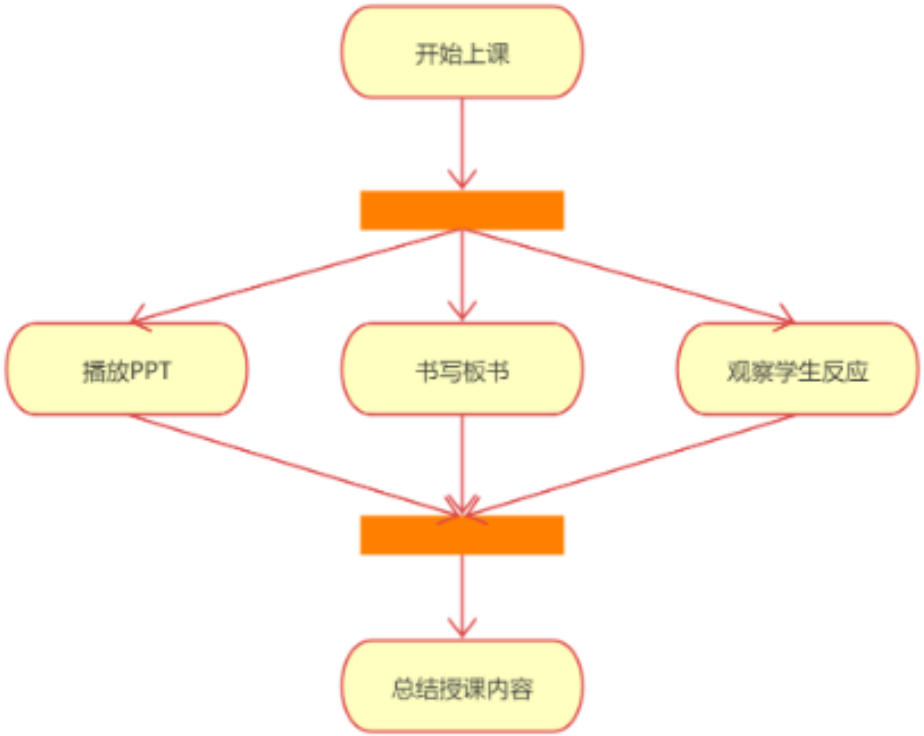
- 步骤合并
  - 菱形也表示多个步骤合并为一条路径，或者多个变量输入到一个步骤的过程

```
…
if(a == 1)
    bVal = true;
else if(b == 2)
    bVal = true;

If(bVal)
    callFun();
else
    printf(“fail“,);
…
```



活动图

- 并行与汇合
  - 系统中存在着在同一时刻，有两个或两个以上的并发控制流的情况
  - 分叉是把控制流分解成两个或多个并发的控制流，汇合表示两个或多个并发控制流在此取得同步



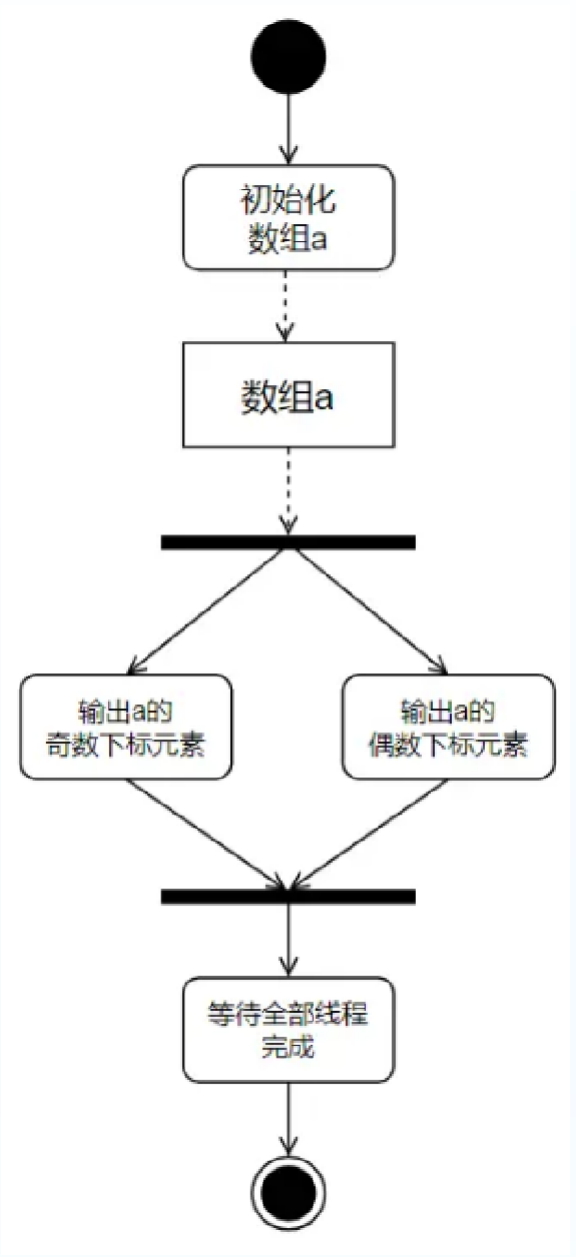
活动图

- 并行与汇合

…
线程1读取数组a的奇数位

线程2读取数组a的偶数位

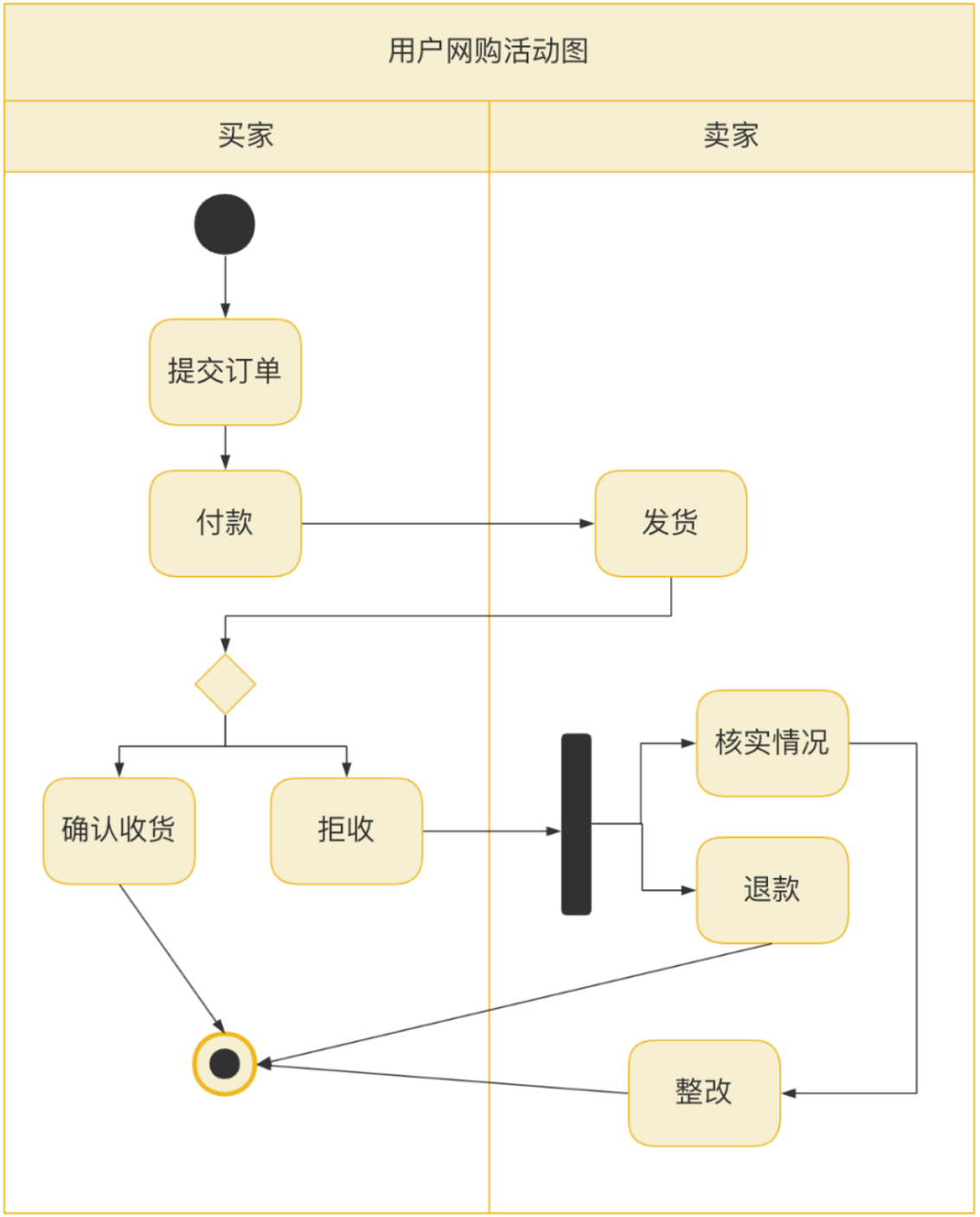
读取完毕后，输出数组a
…



活动图

- 泳道
  - 清楚的表达这些活动或动作是由谁来完成的
  - 每个泳道都以唯一的对象的名称或活动者的名称来命名
  - 活动或动作位于泳道内，不可以跨越泳道，而活动的转移可以跨越泳道

- 泳道



本章小结

- 程序设计基础回顾
- gcc与gdb
- 程序运行原理
- 流程图与序列图

谢谢！

中国科学技术大学
University of Science and Technology of China