C++的变量、常量、程序空间

華中科技大學

- 计算机内存
- 程序内存空间
- 变量
- 数据空间

华中科技大学 计算机学院

金良海 2022.09

1. 内存 (1)

- 计算机的主存贮器是用来存放程序和数据的。
- 主存贮器是以字节为单位的，每个字节能存贮8位二进制数据。
- 80X86的主存贮器一般是由内存条组成的。

内存条：

32K
640K
1M
128M
512M
1G
……

1. 内存 (2)

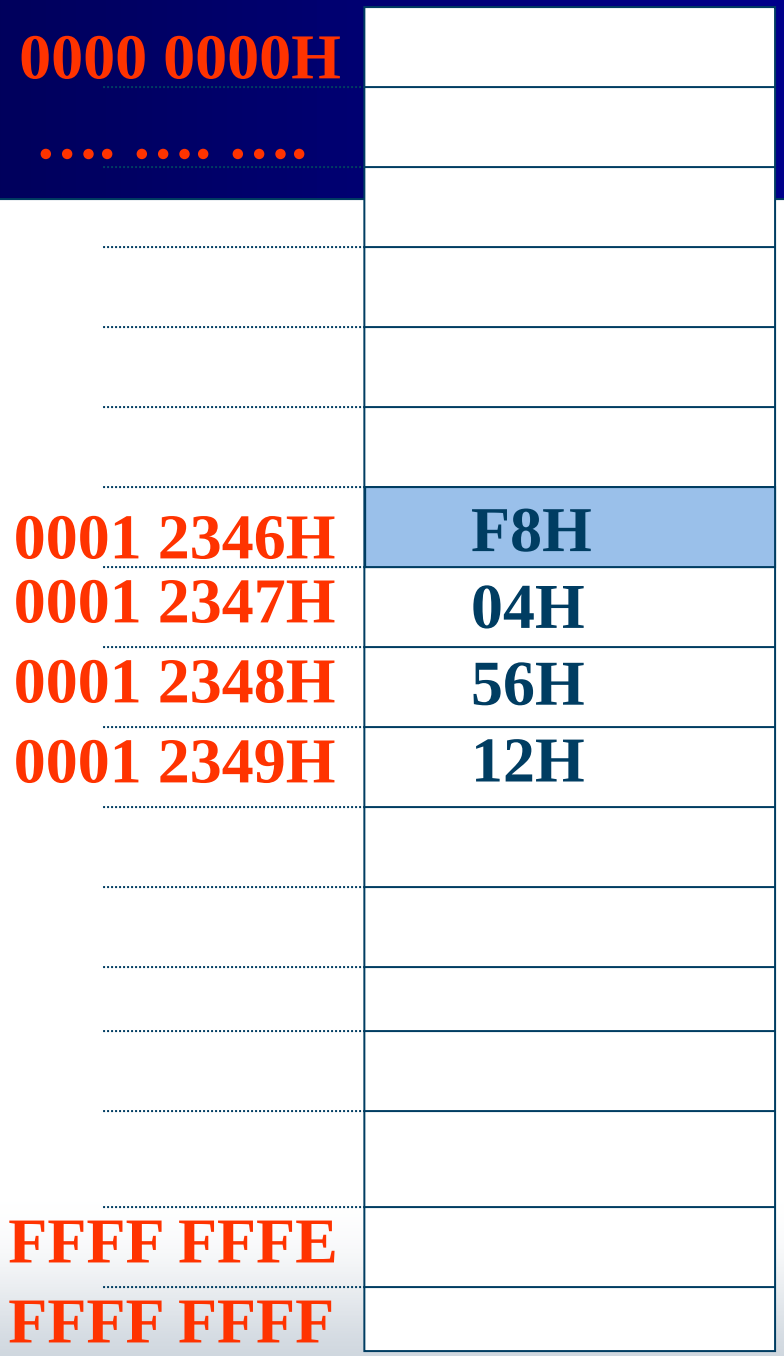
- 为了访问内存中的每个存贮单元（字节），需要给每个存贮单元一个编号，这就是**地址**。
- 内存的地址一般从0开始编号，编号是连续的。例如，1M内存表示有1M (1024*1024)个字节，其地址范围为 00000 ~ FFFFF (16进制)。对于win32 (x86)，能管理4G内存，地址范围为：0000 0000 ~ FFFF FFFF。

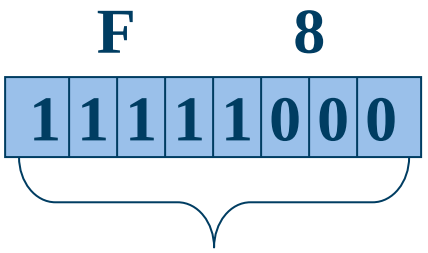
1. 内存 (3)

華中科技大學

每一个字节都有一个地址

字节是最小的寻址单位





8个位组成一个字节 BYTE

Q: 1M字节内存，地址编码需要多少二进制位？

Q:32位地址对应的内存大小可达到多大？

2. 程序内存空间 (1)

- 正在运行中的程序所占用的内存空间：程序代码指令空间、变量空间、常量空间、程序动态申请的空间 (new、malloc等)。
- 这些空间被组织 (管理)为段:
  - 代码段 (用于存放程序的指令代码)
  - 堆栈段 (非静态局部变量、及其他用途)
  - 数据段 (用于全局变量和静态变量)
  - 常量段 (用于保存常量数据, 例如常量字符串)

段名	存储属性	内存分配
代码段 .text	存放可执行程序的指令，存储态和运行态都有	静态
数据段 .data	存放已初始化（非零初始化的全局变量和静态局部变量✦）的数据，存储态和运行态都有	静态
bss段 .bss	存放未初始化（未初始化或者0初始化的全局变量和静态局部变量）的数据，存储态和运行态都有	静态
堆 heap	动态分配内存，需要通过malloc手动申请，free手动释放，适合大块内存。容易造成内存泄漏和内存碎片✦。运行态才有。	动态
栈 stack	存放函数局部变量和参数以及返回值，函数返回后，由操作系统立即回收。栈空间不大，使用不当容易栈溢出✦。运行态才有	静态

2. 程序内存空间 (2)

- 程序动态申请的空间 (new、malloc等)，需要显式用指令去释放。
- 程序正常退出 (main()中的return)，程序的所有空间(代码空间、变量和常量空间) 都会被自动释放。
- 程序非正常退出 (exit)，全局变量和静态变量都会被自动释放，而局部非静态变量不会被释放。
- 程序非正常退出 (abort)，所有变量都不会被释放。
- 不管程序以何种方式退出，代码段空间和常量段空间都会被释放。

3. 变量 (1)

- 高级语言中的任何变量，编译器都会给它分配内存单元，内存单元的大小由变量类型决定。(变量 = 内存单元)

```
char c;        //给c分配1个字节的内存单元
short x;       //给x分配2个字节的内存单元
int y;         //给y分配4个字节的内存单元
int a[10];     //给a分配40个字节的内存单元
int b[3][5];   //给b分配60个字节的内存单元
```

3. 变量 (2)

- **指针。**指针（不管多少重指针）也是变量，所以，编译器会给指针变量分配内存单元。
- 指针变量是用来保存地址的，对于win32 (x86) 系统，地址是32位 (4个字节)，所以编译器会给指针变量分配4个字节的内存单元。
- 不管多少重指针、也不管是什么类型的指针，由于指针本质上是一个地址，所以编译器会给不同的指针变量分配相同大小 (4个字节) 的内存单元。

3. 变量 (3)

对于win32：

```
char *p1;   int **p2;
double ***p3;   int (*p4)[100];
```

编译器会给每个指针变量 p1、p2、p3、p4 分配4个字节的内存单元。

```
struct A {
    long  i, j;
    char  buf[10];
} a[10];                    //180个字节的内存单元
struct A *p5 = a;           //4个字节的内存单元
```

3. 变量 (4)


```
short   x[10][20];
short   *y[10][20];     //怎么解释？
short   (*z)[10][20];
```

- x是1个指向10个元素的数组，其中每个元素又包含20个元素，每个元素是short类型。所以x是1个2维short型的数组，x变量占10*20*2个字节的内存。
- y是1个指向10个元素的数组，其中每个元素又包含20个元素，每个元素是1个short类型的指针。所以y是1个2维short *类型的数组，y变量占10*20*4个字节的内存。
- z是1个指针，指向10个元素的数组，其中每个元素又包含20个元素，每个元素是short类型。由于z是1个指针（指向1个2维数组），所以z变量占4个字节的内存。

4. 数据空间 (1)

(1) 局部空间 (堆栈段)

- 用于分配局部变量 (生命期是函数内部):
  - 函数内部定义的非静态变量 (包括非静态const变量)
  - 函数调用时的传入的实参
  - 函数返回的对象
- 在堆栈段定义的变量都会被C++释放:
  - 函数内部的局部变量在函数返回时被释放;
  - 函数调用时的传入的实参和函数返回的对象, 在主程序调用完函数后被释放。

4. 数据空间 (2)

(1) 局部空间 (堆栈段)

```
class A { … };
int  main() {
    const int k;
    A   a;
    a = f(k, a);  //k匹配吗?
/** 在堆栈中申请空间拷贝k
和a，调用完后将堆栈中临
时对象b拷贝到a，最后释放
堆栈中k,a,b的临时空间 **/
    return 0;  //释放a和k空间
}
```

```
A  f(int k, A a) {
    int x;
    const int  j;
    A  b = a;
    return b;
    /** 在堆栈中申请
    临时空间保存b，
    然后释放函数内部
    定义的x, j, b  **/
}
```

4. 数据空间 (3)

(2) 全局空间 (数据段)

- 全局空间上定义的变量 (对象), 生命期是整个程序的生命期。
- 定义在全局空间的变量 (对象) :
  - 非静态全局变量 (包括 const 对象), 不包括全局的 const 简单类型 (如 const int k)
  - 静态变量 (局部和非局部), 不包括 static const 简单类型 (如 static const int k)
  - 用指令申请的空间 (如new、malloc等)

4. 数据空间 (4)

(2) 全局空间 (数据段)

- 全局变量 (非静态)具有唯一性，作用域是整个程序。2个.cpp文件不能定义同名的全局变量。如果1个.cpp文件需要访问定义在另外1个.cpp文件的全局变量时，必须用extern声明；
- 非局部静态变量 (定义在函数外面的静态变量),作用域是当前.cpp文件。2个不同.cpp文件可以定义同名的静态变量。非局部静态变量可以与其他.cpp中的全局变量同名，但访问不到同名的其他.cpp中的全局变量；
- 局部静态变量 (定义在函数内部的静态变量), 作用域是当前函数内部(生命期是整个程序运行期间)。可以定义与非局部静态变量同名的局部静态变量。

4. 数据空间 (5)

(3) 常量空间 (const段)

一些常量的值被存贮在该空间。定义在该空间的常量是不能修改的，若强行修改则会引起程序的奔溃。

定义在const段的3个典型常量 (变量):

- 常量字符串。类型是 const char *，而不是 char *。字符串需要以 0 结尾，所以 “abc” 的长度是3，需要4个字节的存贮空间。
- 所有(全局、局部、类内)的静态const简单变量 (如 static const int k)
- 全局非静态const简单变量 (如 const int k)

4. 数据空间 (6)

(3) 常量空间 (const段)

```
const char *p = “abc”;
char  c1 = p[1];
char  c2 = “abc”[1];              //???
char  c3 = “abc”[-1];             //???
p[0] = ‘1’;                       //???
(char *)p[0] = ‘1’;               //???
char *q = “abc”;                  //???
char *q = (char *)“abc”;          //???
q[0] = ‘1’;                       //???
(char *)“abc”[1] = ‘1’;           //???
char s[20];  strcpy(s, “abc”);
s[3] = ‘1’;                       //???
```

4. 数据空间 (7)


```
const int x = 1;
struct  A {
      int  k;
      const int  i;
      static const int  j;
      A( ): i (-1) { }
} a;
const int  A::j = 2;

int main( ) {
    *(int *)&x = 0;        //语法正确, 程序奔溃
    *(int *)&A::j = 0;     //语法正确, 程序奔溃
    *(int *)&a.i = 0;      //正确, a.i = 0
}
```

想一想, why?

为什么可以修改类的const实例数据成员?

The end.