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

运行时存储空间的组织与管理-I

《编译原理(H)》

张昱

0551-63603804，yuzhang@ustc.edu.cn

中国科学技术大学

计算机科学与技术学院

本章内容

术语

- 过程的活动(activation): 过程的一次执行
- 活动记录

  过程的活动需要可执行代码和存放所需信息的存储空间，后者称为活动记录

本章内容

- 一个活动记录中的数据布局
- 程序执行过程中，所有活动记录的组织方式
- 非局部名字的管理、参数传递方式、堆管理
- 几种典型的编译运行时系统（新增）

影响存储分配策略的语言特征

- 过程能否递归
- 当控制从过程的活动返回时，局部变量的值是否要保留
- 过程能否访问非局部变量
- 过程调用的参数传递方式
- 过程能否作为参数被传递
- 过程能否作为结果值传递
- 存储块能否在程序控制下被动态地分配
- 存储块是否必须被显式地释放

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

6.1 名字、绑定、作用域

- 过程定义、调用、活动
- 名字、绑定、作用域、生存期
- 活动记录的常见布局
  - 字节寻址、类型、次序、对齐
- 同名变量的处理

基本概念：过程/函数

过程(包括函数、方法等)

- 过程定义、过程调用、形式参数、实在参数
- 活动(过程的一次调用)、活动的生存期

形式参数

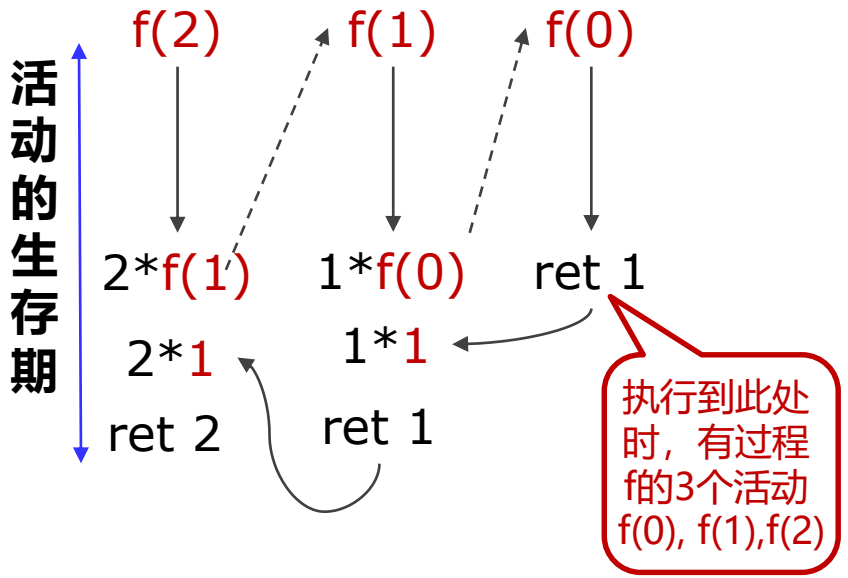
过程定义

```
int  f(int n){
  if (n<0) error("arg<0");
  else if (n==0) return 1;
  else return n*f(n-1);
}

...print(f(2));
```

n-1是实在参数
f(n-1)是过程调用

2是实在参数
f(2)是过程调用



基本概念：名字

名字≡ 标识符

- 符号的文本表示，用来引用变量、常量、函数、类型等实体
- 构词规则（语言的硬性规定）、命名约定（软性约束）
- 关键字和保留字：相同的名字可否有不同的含义？

Fortran：关键字不是保留字

```
Integer Apple
Integer = 4
Integer Real
Real Integer
```

关键字可以作为变量名

C/C++、Java：关键字是保留字

```
int i; /* 合法 */
float int;  /*不合法*/
```

基本概念：绑定

- **绑定(binding)**

  程序中实体(如变量、常量、函数、类型等)和属性(如类型、值、存储位置、作用域等)的关联

  - **变量**和其**类型、值**
  - **符号**和其**操作**：如“+”绑定到整数加add、浮点加fadd(x86)/adf(ARM)

- **绑定时间**：绑定被创建的时间点

  - 语言设计时：程序结构、可能的类型
  - 语言实现时：I/O、运算的溢出、类型等价性
  - 程序编写时：算法、名字
  - 编译时：数据布局的规划
  - 链接时：整个程序在内存中的布局
  - 加载时：物理地址的选择
  - 运行时：变量-值的绑定、程序启动时间、过程进入时间、语句执行时间等

基本概念：绑定

绑定时机

语言设计时、语言实现时、程序编写时、编译时、链接时、加载时、运行时

如：count = count + 2;

- count的类型在编译时绑定
- count的可能取值集合在语言设计时绑定
- count的值在运行时绑定
- +的含义在编译时当确定操作数的类型时被绑定
- 2的内部表示在语言设计时被绑定

基本概念：变量

- 变量：程序语言中对机器的内存单元的抽象
  - 名字、类型、字宽、地址、作用域、生存期(lifetime)
  - ① 绝大多数的变量都有**名字**
  - 没有名字的变量：临时变量、存储在堆中的变量
  - ② 变量的**地址** ≡ 左值 变量的**值** ≡ 右值
  - 程序中相同的变量在不同时间关联到不同的地址
  - 环境把名字映射到左值，而状态把左值映射到右值
  - 赋值改变状态，但不改变环境
  - 过程调用改变环境：不同的活动有不同的活动记录
  - 如果环境将名字 x 映射到存储单元 s，则说 x 被绑定

基本概念：作用域

作用域（scope）

- **规定了一个名字在程序中的可见范围**
- **静态作用域（词法作用域）：在编译时确定**
  - 由代码的静态结构决定，即通过程序源代码中的层次结构（如函数、块）来确定
  - 查找变量时，编译器会根据函数或语句块的嵌套关系，从内向外逐层查找变量的定义
  - **常见于**：C、C++、Java、Python、JavaScript 等大多数现代编程语言
- **动态作用域：在程序运行时确定**
  - 变量的作用域取决于函数调用的顺序
  - 查找变量时，程序会根据调用栈的顺序从调用者往回逐层查找变量的定义
  - **常见于**：某些早期的语言如 Lisp的早期版本以及一些脚本语言

基本概念

静态概念和动态概念的对应

静 态 概 念	动 态 对 应
过程的定义	过程的活动
名字的声明	名字的绑定
声明的作用域	绑定的生存期

基本概念

活动记录（activation record）

常见布局 →

- 临时数据：存放中间结果
- 局部数据：过程的局部变量
- 机器状态：保存过程调用前的机器状态信息
- 访问链：用于访问非局部数据 —— C语言无须此域
- 控制链：用来指向调用者的活动记录
- 参数：存放调用者提供的实在参数
- 返回值：用于存放被调用者返回给调用者的值

局部数据的布局

存储布局的一些因素

- 字节是可编址内存的最小单位
- 变量所需的存储空间可以根据其类型而静态确定
- 一个过程所声明的局部变量，按这些变量声明时出现的次序，在局部数据域中依次分配空间
- 局部数据的地址可以用相对于活动记录中某个位置的地址来表示
- 数据对象的存储布局还需考虑对齐问题

对齐对存储size的影响

例 在SPARC/Solaris工作站上下面两个结构体的size分别是24和16，为什么不一样？

```
typedef struct  _a{          typedef struct  _b{
    char    c1;                      char c1;
    long    i;                       char    c2;
    char    c2;                      long i;
    double f;                        double f;
}a;                          }b;
```

对齐：char : 1, long : 4, double : 8

对齐对存储size的影响

例 在SPARC/Solaris工作站上下面两个结构体的size分别是24和16，为什么不一样？

```
typedef struct  _a{          typedef struct  _b{
    char    c1;     0            char c1;      0
    long    i;      4            char   c2;    1
    char    c2;     8            long i;       4
    double f;       16           double f;     8
}a;                              }b;
```

对齐：char : 1, long : 4, double : 8

对齐对存储size的影响

例 在X86/Linux工作站上下面两个结构体的size分别是20和16，为什么不一样？

```
typedef struct  _a{          typedef struct  _b{
    char    c1;     0            char c1;      0
    long    i;      4            char   c2;    1
    char    c2;     8            long i;       4
    double f;       12           double f;     8
}a;                              }b;
```

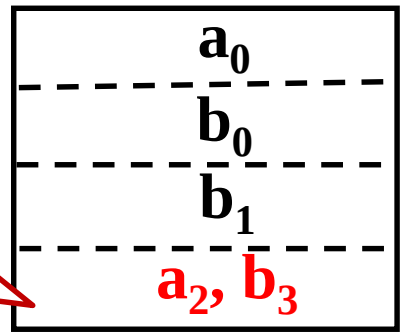
对齐：char : 1, long : 4, double : 4

程序块与同名变量的处理

- 现代语言一般可在**程序块**中的任何地方声明变量
  - C99、C++、 Java: 作用域为从声明处开始到该语句块结尾结束
  - C#：作用域为整个语句块
- **C/C++中可以嵌套声明同名变量，按最近(小)嵌套作用域规则，但在Java和C#中不合法—容易出错**
- **并列程序块**不会同时活跃,不同并列块中的变量可以**重叠分配**

a_i: 作用域B_i中声明的变量a

B_2、B_3不会同时运行，故B_2中的a_2和B_3中的b_3复用存储空间




```
main()
{
  int a = 0;
  int b = 0;
  {
    int b = 1;
    {
      int a = 2;
    }
    {
      int b = 3;
    }
  }
}
```

B_0 B_1 B_2 B_3

全局作用域

- C、C++、Python等**允许在函数定义外声明变量**
- C、C++有全局变量**声明和定义**，后者要分配内存单元
- C、C++同名局部变量与全局变量作用域重叠的，**重叠部分按局部变量处理**

C++

```
#include <iostream>
void func( float );
const int a = 17;            // global constant
int b, c;                    // global variable
int main()
{
        b = 4;               // assignment to global b
        c = 6;               // assignment to global c
        func(42.8);  return 0;
}
void func( float c)       // prevents access to global c
{
        float b;             // prevent access to global b
        b = 2.3;             // assignment to local b
        cout << " a = " << a;      // output global a (?)
        cout << " b = " << b;      // output local b (?)
        cout << " c = " << c;      // output local c (?)
}
```

Output:
A = 17 b = 2.3 c = 42.8

Python：全局变量可以在函数中引用，但是只能对在函数中声明为global的全局变量赋值

```
a = 3
def Fuc():
  print (a)
  a = a + 1
Fuc()
```

```
a = 3
def Fuc():
  print (a)
Fuc()
```

```
a = 3
def Fuc():
  global a
  print (a)
  a = a + 1
Fuc()
```

类与作用域、命名空间

类有自己的局部变量

- 类变量、实例变量
- 方法中声明的变量
- 访问属性
  - public
  - protected
  - private

```
namespace std
 {
      ..
      int abs (int );
      ..
 }
```

命名空间

- C++允许用户创建自己的命名作用域
- 如标准的cstdlib头文件包含一些库函数的原型声明

```
#include <cstdlib>
 int main()
 {
     int alpha;
     int beta;
     ..
     alpha = std::abs(beta);
  }
```

Scope resolution operator

例　题　1

一个C语言程序及其在X86/Linux操作系统上的编译结果如下。根据生成的汇编程序来解释程序中四个变量的存储分配、生存期、作用域和置初值方式等方面的区别

```
static long aa = 10;
short bb = 20;
extern int f( );
int func( ) {
    static long cc = 30;
    short dd = 40;
    cc = f(cc,dd);
}
```

在64位系统用gcc –S编译


```
.data
.align 8
    .type aa,@object
    .size aa,8
aa:
    .quad 10
    .globl bb
    .align 2
    .type bb,@object
    .size bb,2
bb:
    .value 20
```

分配8字节

```
static long aa = 10;
short bb = 20;
extern int f();
```

```
    .align 8
    .type    cc.1797,@object
    .size    cc.1797, 8
cc.1797:
    .quad 30
.text
    .globl func
    .type func, @function
func: . . .
    movw $40,-2(%rbp)
    movswl -2(%rbp), %edx
    movq cc.1797(%rip), %rax
    movl  %edx, %esi
    movq %rax, %rdi
    movl  $0, %eax
    call  f
```

```
int func( ) {
  static long cc = 30;
  short dd = 40;
  cc = f(cc, dd);}
```

在64位系统用gcc –S编译


```
.data
.align 8
    .type aa,@object
    .size aa,8
aa:
    .quad 10
    .globl bb
    .align 2
    .type bb,@object
    .size bb,2
bb:
    .value 20
```

分配8字节

```
static long aa = 10;
short bb = 20;
extern int f();
```

```
    .align 8
    .type    cc.1797,@object
    .size    cc.1797, 8
cc.1797:
    .quad 30
.text
    .globl func
    .type func, @function
func: . . .
    movw $40,-2(%rbp)
    movswl -2(%rbp), %edx
    movq cc.1797(%rip), %rax
    movl  %edx, %esi
    movq %rax, %rdi
    movl  $0, %eax
    call  f
```

```
int func( ) {
  static long cc = 30;
  short dd = 40;
  cc = f(cc, dd);}
```

在64位系统用gcc –S编译


```
.data
.align 8
    .type aa,@object
    .size aa,8
aa:
    .quad 10
    .globl bb
    .align 2
    .type bb,@object
    .size bb,2
bb:
    .value 20
```

```
static long aa = 10;
short bb = 20;
extern int f();
```

```
    .align 8
    .type    cc.1797,@object
    .size    cc.1797, 8
cc.1797:
    .quad 30
.text
    .globl func
    .type func, @function
func: . . .
    movw $40,-2(%rbp)
    movswl -2(%rbp), %edx
    movq cc.1797(%rip), %rax
    movl  %edx, %esi
    movq %rax, %rdi
    movl  $0, %eax
    call  f
```

```
int func( ) {
  static long cc = 30;
  short dd = 40;
  cc = f(cc, dd);}
```

在64位系统用gcc –S编译


```
.data
.align 8
    .type aa,@object
    .size aa,8
aa:
    .quad 10
    .globl bb
    .align 2
    .type bb,@object
    .size bb,2
bb:
    .value 20
```

```
static long aa = 10;
short bb = 20;
extern int f();
```

```
    .align 8
    .type   cc.1797,@object
    .size   cc.1797, 8
cc.1797:
    .quad 30
.text
    .globl func
    .type func, @function
func: . . .
    movw $40,-2(%rbp)
    movswl -2(%rbp), %edx
    movq cc.1797(%rip), %rax
    movl  %edx, %esi
    movq %rax, %rdi
    movl  $0, %eax
    call  f
```

```
int func( ) {
  static long cc = 30;
  short dd = 40;
  cc = f(cc, dd);}
```

在64位系统用gcc –S编译


```
.data
.align 8
    .type aa,@object
    .size aa,8
aa:
    .quad 10
    .globl bb
    .align 2
    .type bb,@object
    .size bb,2
bb:
    .value 20
```

```
static long aa = 10;
short bb = 20;
extern int f();
```

```
    .align 8
    .type   cc.1797,@object
    .size   cc.1797, 8
cc.1797:
    .quad 30
.text
    .globl func
    .type func, @function
func: . . .
    movw $40,-2(%rbp)
    movswl -2(%rbp), %edx
    movq cc.1797(%rip), %rax
    movl  %edx, %esi
    movq %rax, %rdi
    movl  $0, %eax
    call  f
```

```
int func( ) {
  static long cc = 30;
  short dd = 40;
  cc = f(cc, dd);}
```

在64位系统用gcc –S编译


```
.data
.align 8
    .type aa,@object
    .size aa,8
aa:
    .quad 10
    .globl bb
    .align 2
    .type bb,@object
    .size bb,2
bb:
    .value 20
```

```
static long aa = 10;
short bb = 20;
extern int f();
```

```
    .align 8
    .type    cc.1797,@object
    .size    cc.1797, 8
cc.1797:
    .quad 30
.text
    .globl func
    .type func, @function
func: . . .
    movw $40,-2(%rbp)
    movswl -2(%rbp), %edx
    movq cc.1797(%rip), %rax
    movl  %edx, %esi
    movq %rax, %rdi
    movl  $0, %eax
    call  f
```

```
int func( ) {
  static long cc = 30;
  short dd = 40;
  cc = f(cc, dd);}
```

在64位系统用gcc –S编译


```
.data
.align 8
    .type aa,@object
    .size aa,8
aa:
    .quad 10
    .globl bb
    .align 2
    .type bb,@object
    .size bb,2
bb:
    .value 20
```

```
static long aa = 10;
short bb = 20;
extern int f();
```

实参dd先提升成4字节整型，

再通过寄存器esi传参

```
    .align 8
    .type   cc.1797,@object
    .size   cc.1797, 8
cc.1797:
    .quad 30
.text
    .globl func
    .type func, @function
func: . . .
    movw $40,-2(%rbp)
    movswl -2(%rbp), %edx
    movq cc.1797(%rip), %rax
    movl  %edx, %esi
    movq %rax, %rdi
    movl  $0, %eax
    call  f
```

```
int func( ) {
  static long cc = 30;
  short dd = 40;
  cc = f(cc, dd);}
```

在64位系统用gcc –S编译


```
.data
.align 8
    .type aa,@object
    .size aa,8
aa:
    .quad 10
    .globl bb
    .align 2
    .type bb,@object
    .size bb,2
bb:
    .value 20
```

```
static long aa = 10;
short bb = 20;
extern int f();
```

实参cc先加载到寄存器rax，

再通过寄存器rdi传参

```
    .align 8
    .type   cc.1797,@object
    .size   cc.1797, 8
cc.1797:
    .quad 30
.text
    .globl func
    .type func, @function
func: . . .
    movw $40,-2(%rbp)
    movswl -2(%rbp), %edx
    movq cc.1797(%rip), %rax
    movl  %edx, %esi
    movq %rax, %rdi
    movl  $0, %eax
    call  f
```

```
int func( ) {
  static long cc = 30;
  short dd = 40;
  cc = f(cc, dd);}
```

在64位系统用gcc –S编译


```
func:
        pushq   %rbp
        movq    %rsp, %rbp
        subq    $16, %rsp
        movw    $40, -2(%rbp)
        movswl  -2(%rbp), %edx
        movq    cc.1797(%rip), %rax
        movl    %edx, %esi
        movq    %rax, %rdi
        movl    $0, %eax
        call    f
        cltq
        movq    %rax, cc.1797(%rip)
        nop
        leave
        ret
```

分配局部变量空间，按16字节对齐

```
int func( ) {
  static long cc = 30;
  short dd = 40;
  cc = f(cc, dd);
}
```

f 函数的返回值通过寄存器eax 返回

cltq等效于movslq %eax, %rax

中国科学技术大学

University of Science and Technology of China

下期预告：活动记录的组织