Universitetet i Oslo
Institutt for Informatikk

Reliable Systems
Martin Steffen

# INF 5110: Compiler construction

Spring 2024

## Series 7

22. 4. 2024

**Topic: Run-time environments (Chapter 7)**

**Issued: 22. 4. 2024**

**Exercise 1 (Run-time environment)** Draw a possible organization for the runtime environment of the following C program, for the following two situations. See corresponding figures from the lecture as inspiration (for example, the slide entitled "Stack gcd", approximately at slide 8.18):

1. after entry into block `A` in function `f`.
2. after entry into block `B` in function `g`.

```
1  int a[10];
2  char * s = "hello";
3
4  int f(int i, int b[])
5  { int j=i;
6   A:{ int i=j;
7       char c = b[i];
8       //...;
9     }
10    return 0;
11 }
12
13 void g(char * s)
14 { char c = s[0];
15  B:{ int a[5];
16      // ...;
17    }
18 }
19
20 main ()
21 { int x=1;
22   x = f(x,a);
23   g(s);
24   return 0;
25 }
```

**Exercise 2 (Activation records (Pascal))** Draw the stack of activation reords for the following *Pascal* program, showing the *control* and *access* links, after the second call to procedure `c`. Describe how the variable `x` is accessed from within `c`.

```
program env;

procedure a;
var x: integer;

    procedure b;
        procedure c;
        begin
            x := 2;
            b;
        end;
    begin (* b *)
        c;
    end;

begin (* a *)
    b;
end;

begin (* main *)
    a;
end.
```

**Exercise 3 (Access chaining vs. display)** An alternative to access chaining in a language with local procedures is to keep the access links in an array *outside* the stack, *indexed* by the *nesting level*. This array is called the *display*. For example, the run-time stacks of the program `chain` and the corresponding stack picture on the slide entitled "access chaining" at approx. slide 8-36 from the lecture would now look as Figure 1 resp. Figure 2.

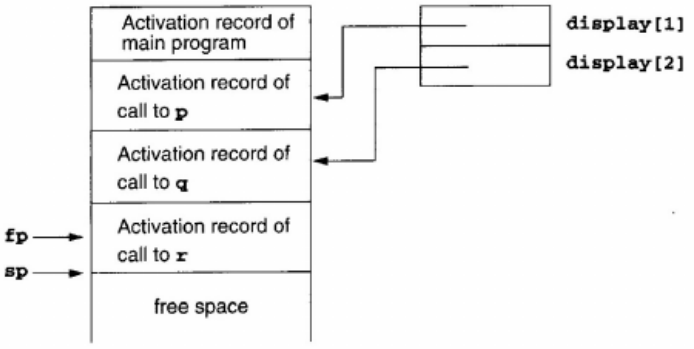


Figure 1: RTE with display (1)

1. Describe how a display can improve *efficiency* of nonlocal references from deeply nested procedures.

2. Redo the previous Exercise 2 from this sheet, using a *display*.

**Exercise 4 (Virtual function tables and memory layout for classes)** Draw the memory layout of objects of the following C$^{++}$ classes, together with the *virtual function tables*.

```
class A
{ public:
```

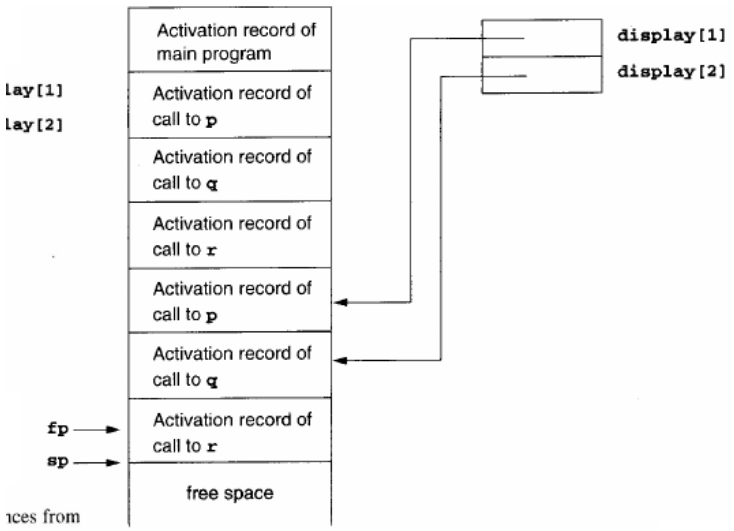


Figure 2: RTE with display (2)

```
3     int a;
4     virtual void f();
5     virtual void g();
6   };
7
8   class B : public A
9   { public:
10    int b;
11    virtual void f();
12    void h();
13  };
14
15
16  class C: public B
17  { public:
18    int c;
19    virtual void g();
20  }
```

**Exercise 5 (Parameter passing)** Give the output of the following program (written in C syntax) using the 4 parameter passing methods discussed in in the lecture.

```
1  #include <stdio.h>
2  int i = 0;
3
4  void p(int x, int y)
5  { x += 1;
6    i += 1;
7    y += 1;
8  }
9
10 main ()
11 { int a[2] = {1,1};
12   p(a[i],a[i]);
13   printf("%d-%d\n",a[0], a[1]);
14   return 0;
15 }
```

**Exercise 6 (Parameter passing)** Give the output of the following program (written in C syntax) using the 4 parameter passing methods discussed in the lecture.

```c
#include <stdio.h>
int i = 0;

void swap (int x, int y)
{
  x = x + y;
  y = x - y;
  x = x - y;
}

main ()
{ int a[3] = {1,2,0};
  swap(i,a[i]);
  printf("%d-%d-%d-%d\n",i,a[0], a[1],a[2]);
  return 0;
}
```