Chapter 7
Runtime Environments

한양대학교 컴퓨터공학부

컴파일러

2014년 2학기

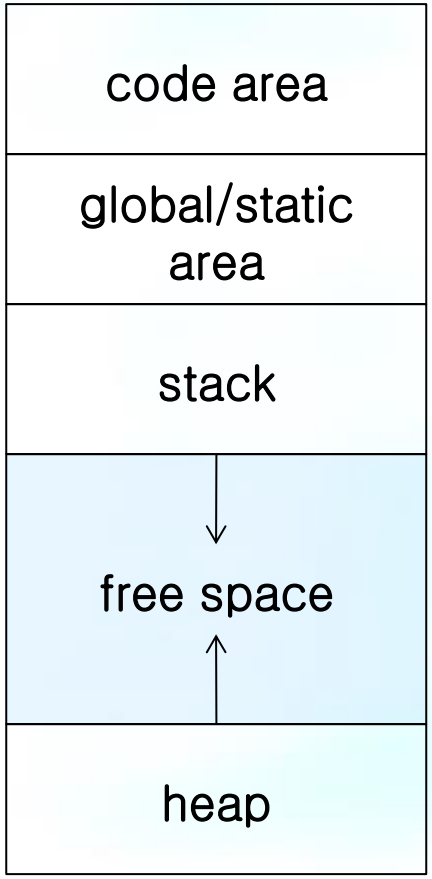
Introduction

- So far, the analysis depends only on the properties of the source language
- From this chapter, how a compiler generates executable code
  - Dependent on the details of the target machine
- **runtime environment**
  - structure of the target computer's registers and memory
- three kinds of environments
  - fully static environment → FORTRAN77 변수 미리 다 할당, 수학계산,일기예보
  - stack-based environment → C, C++, Pascal, Ada
  - fully dynamic environment → Lisp garbage collector가 변수 관리

the runtime environments

- Issues
  - Scoping and allocation issues
  - Nature of procedure calls
  - Varieties of parameter passing mechanisms
- Keep in mind
  - A compiler can maintain a environment only indirectly
    - must generate code to perform the necessary maintenance operations during program execution
  - an interpreter can maintain the environment directly

Memory Organization



- code: cannot make changes to the code area during execution
- global/static data: can be fixed in memory prior to execution
- dynamic data: stack and heap

procedure activation record

- also called a **stack frame**
- special-purpose registers to keep track of execution
  - program counter (pc)
  - stack pointer (sp)
  - frame pointer (fp)
  - argument pointer (ap)

7.3 Stack-based Runtime Environments

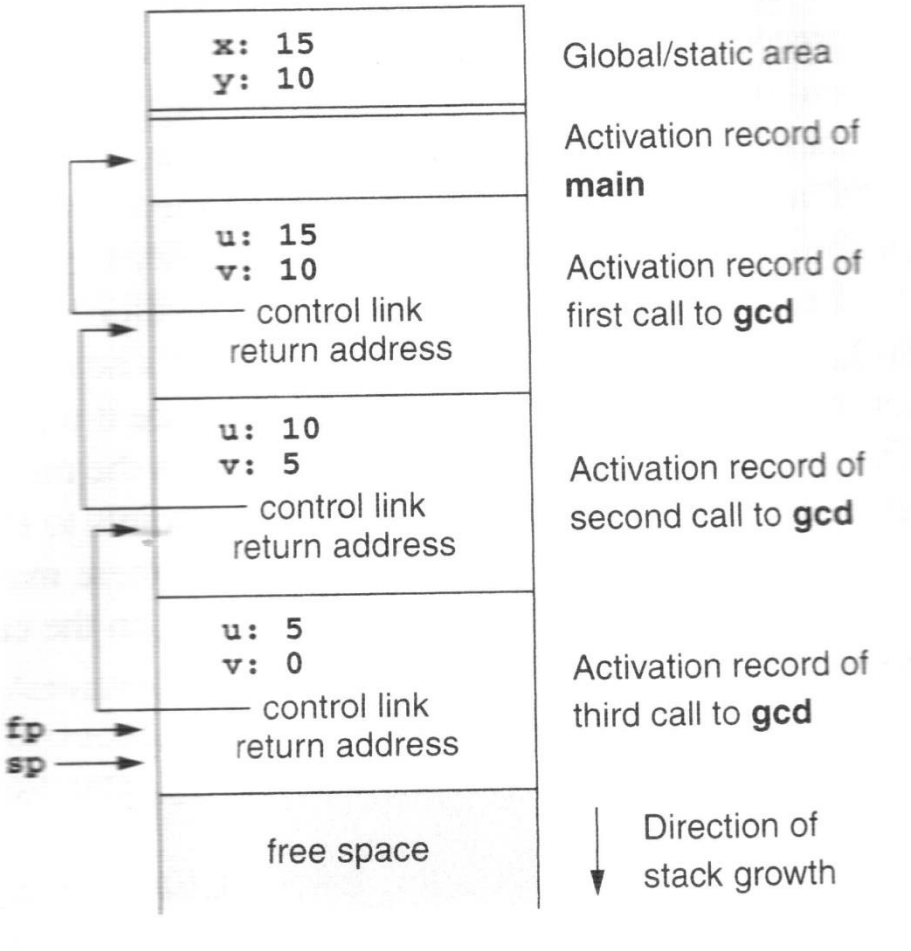
- frame pointer
  - usually kept in a register
- control link
  - dynamic link
  - old fp
- stack pointer
  - top of stack

stack-based environments without local procedures

- fig. 7.3

```
#include <stdio.h>

int x,y;

int gcd( int u, int v)
{ if (v == 0) return u;
  else return gcd(v,u % v);
}

main()
{ scanf("%d%d",&x,&y);
  printf("%d\n",gcd(x,y));
  return 0;
}
```

- fig. 7.4

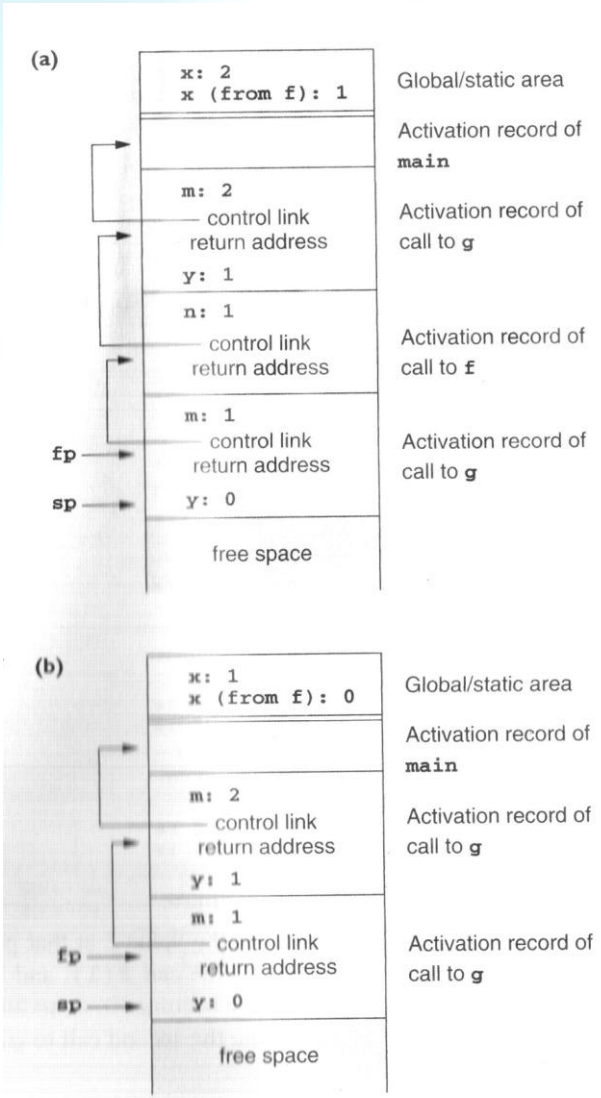


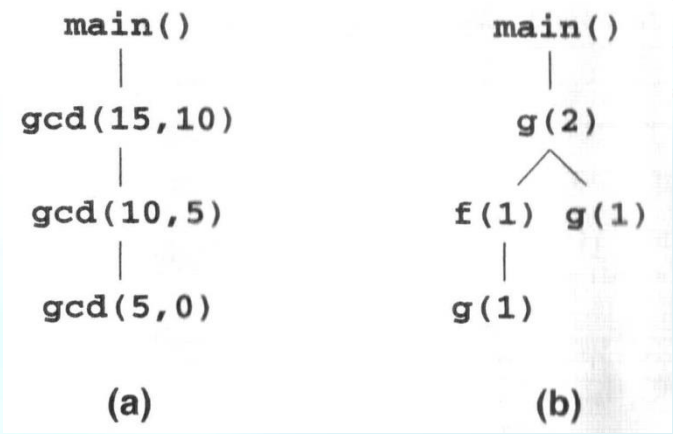
Example 7.3

- fig. 7.5

```
int x = 2;

void g(int); /* prototype */

void f(int n)
{ static int x = 1;
  g(n);
  x--;
}

void g(int m)
{ int y = m-1;
  if (y > 0)
  { f(y);
    x--;
    g(y);
  }
}


main()
{ g(x);
  return 0;
}
```

Example 7.3

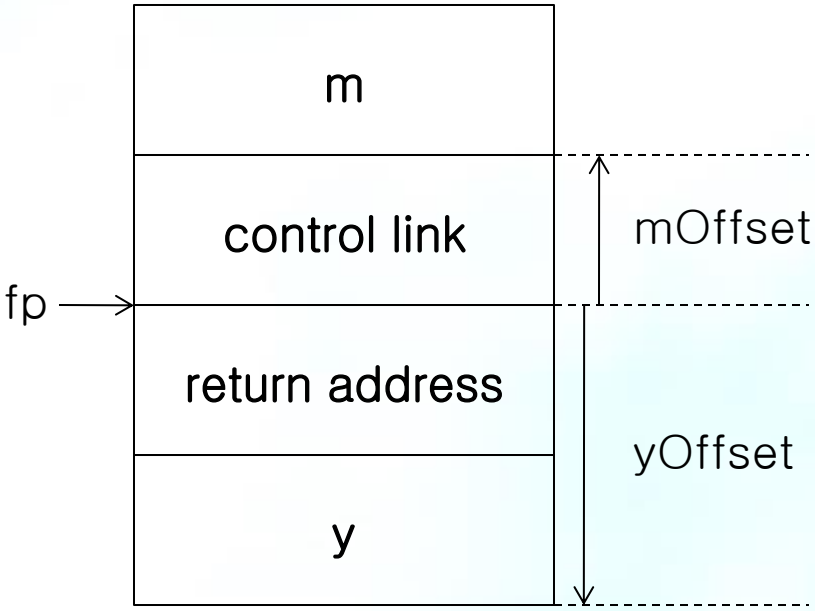




char y[10]
gets[y] = y가 10 byte가 넘으면 return address 변경 될 수 있다.
<stack buffer overflow>

access to names

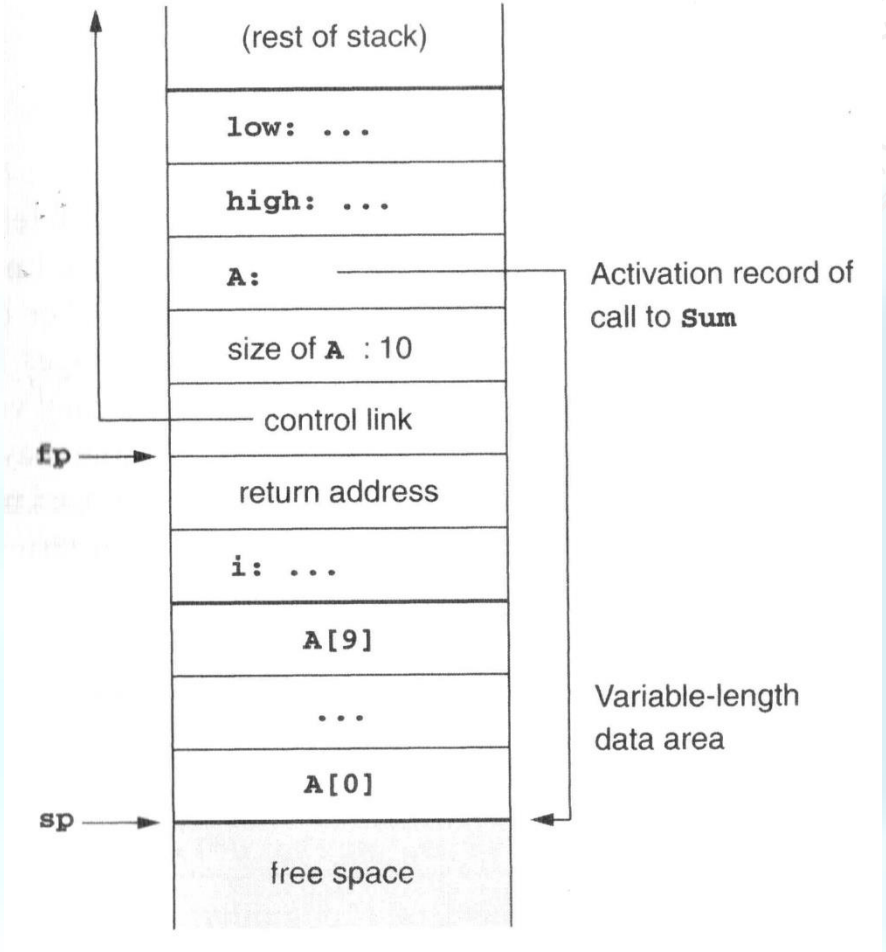
- m = 4(fp)
- y = -6(fp)



dealing with variable-length data

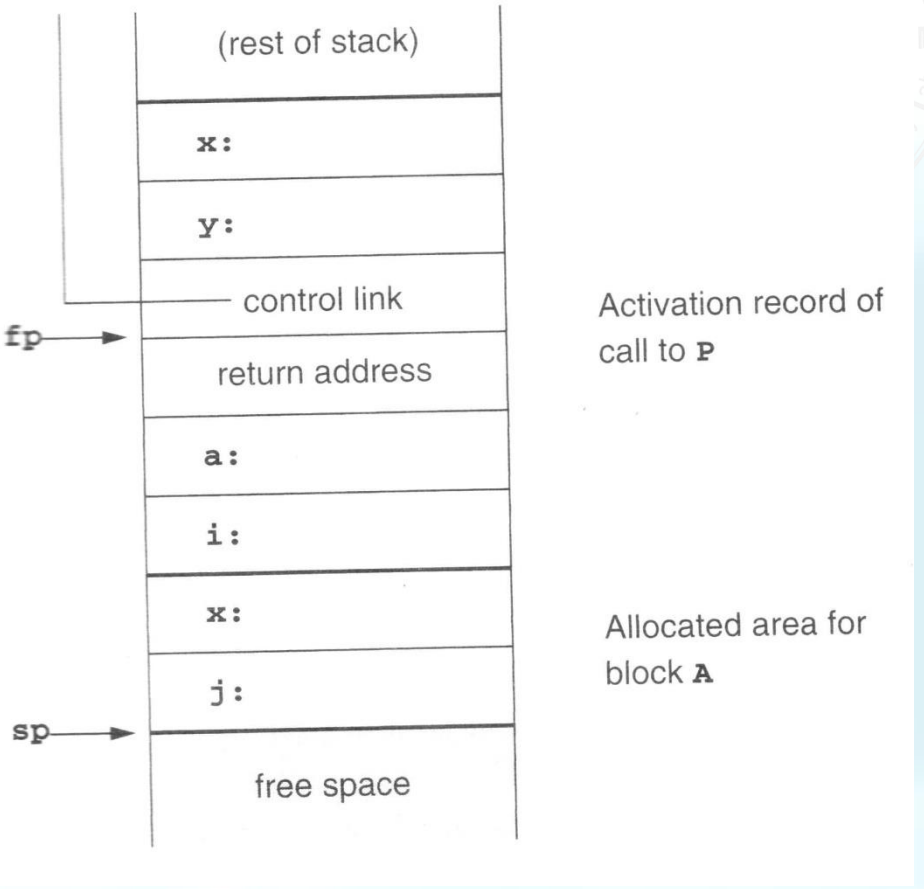
- p. 362

```
type Int_Vector is
      array(INTEGER range <>) of INTEGER;

function Sum (low,high: INTEGER;
                  A: Int_Vector) return INTEGER
is
   i: INTEGER;
begin
  ...
end Sum;
```



local temporaries and nested declarations


```
void p( int x, double y)
{ char a;
   int i;
   ...
A:{ double x;
     int j;
     ...
   }
   ...
B:{ char * a;
     int k;
     ...
   }
   ...
}
```

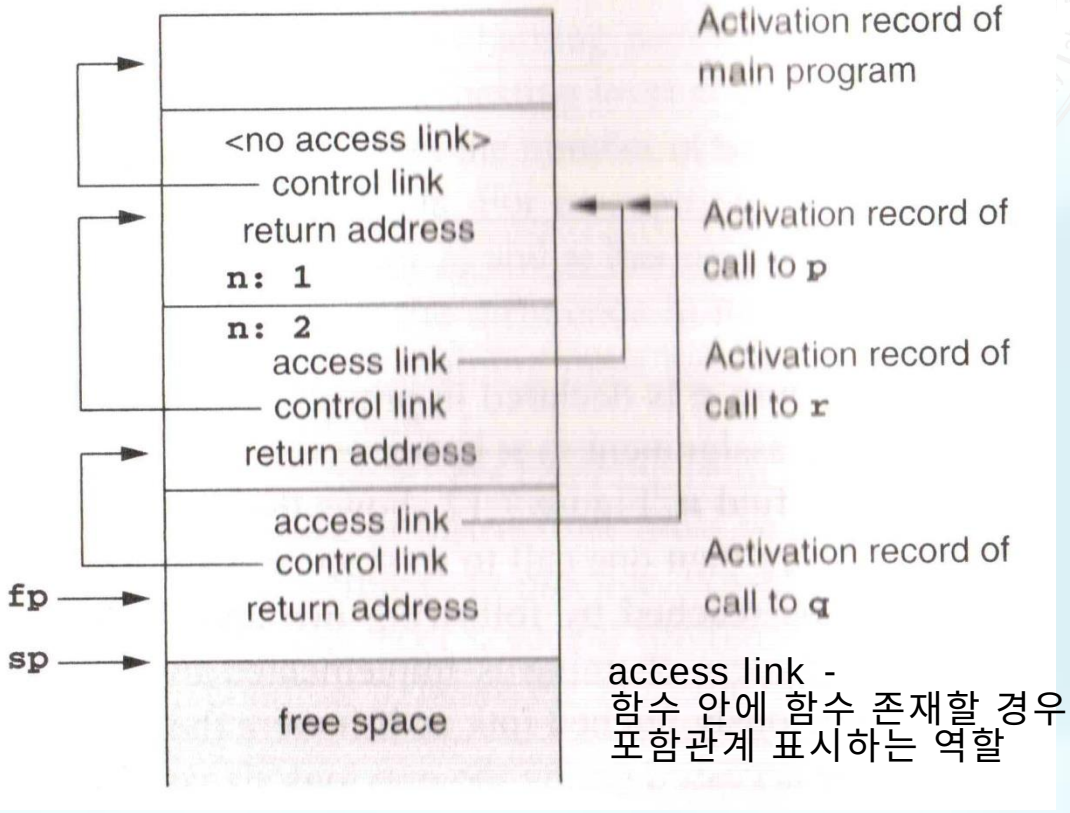


Stack-based environments with local procedures

- Fig. 7.8

```
program nonLocalRef;

procedure p;
var n: integer;

      procedure q;
      begin
        (* a reference to n is now
           non-local non-global *)
      end; (* q *)

      procedure r(n: integer);
      begin
        q;
      end; (* r *)

begin (* p *)
  n := 1;
  r(2);
end; (* p *)

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

- Fig. 7.10



Parameter Passing Mechanisms

- pass by value c
  - no special effort on the part of the compiler
- pass by reference (C++)

```
void inc2(int &x)
{ ++x; ++x; }
```

- pass by value-result (Ada)

```
void p(int x, int y)        a=2
{ ++x; ++y; }
main()
{ int a=1;
  p(a, a);
  return 0; }
```

- pass by name (Algol60)

```
int i; int a[10];
void p(int x)
{ ++i; ++x; }
main ()
{ i = 1;                    a[2] =3으로 변경됨
  a[1] = 1;
  a[2] = 2;
  p(a[i]);
  return 0;
}
```