Intermediate Code

"Abstract" code generated from AST

Motivation for use: **Simplicity and Portability**

- Machine independent code.
- Enables common optimizations on intermediate code.
- Machine-dependent code optimizations postponed to last phase.

Intermediate Forms

- Stack machine code:
  Code for a "postfix" stack machine.
- Two address code:
  Code of the form "`add` r_1, r_2"
- Three address code:
  Code of the form "`add` $src_1, src_2, dest$"
  Quadruples and Triples: Representations for three-address code.

Quadruples

Explicit representation of three-address code.
Example: `a := a + b * -c;`

Instr	Operation	Arg 1	Arg 2	Result
(0)	`uminus`	`c`		t_1
(1)	`mult`	`b`	t_1	t_2
(2)	`add`	`a`	t_2	t_3
(3)	`move`	t_3		`a`

Triples

Representation of three-address code with implicit destination argument.
Example: `a := a + b * -c;`

Instr	Operation	Arg 1	Arg 2
(0)	`uminus`	`c`	
(1)	`mult`	`b`	(0)
(2)	`add`	`a`	(1)
(3)	`move`	`a`	(2)

Intermediate Forms

Choice depends on convenience of further processing

- Stack code is simplest to generate for expressions.
- Quadruples are most general, permitting most optimizations including code motion.
- Triples permit optimizations such as *common subexpression elimination*, but code motion is difficult.

Runtime Storage Organization

Storage for code and data.

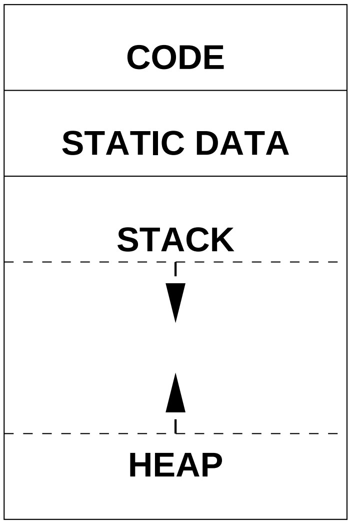
- Code Area: Procedures, functions, methods.
- Static Data Area: "Permanent" data with statically known size.
- Stack: Temporary Data with known lifetime.
- Heap: Temporary Data with unknown lifetime (dynamically allocated).

Issues in Storage Organization

- Recursion
- Block structure and nesting (*nested procedures*).
- Parameter passing (*by value, reference, name*).
- Higher order procedures (*procedures as parameters to other procedures*).
- Dynamic Storage Management (*malloc, free*).

Storage Areas

Storage Organization for a typical procedural language.



CODE

STATIC DATA

STACK

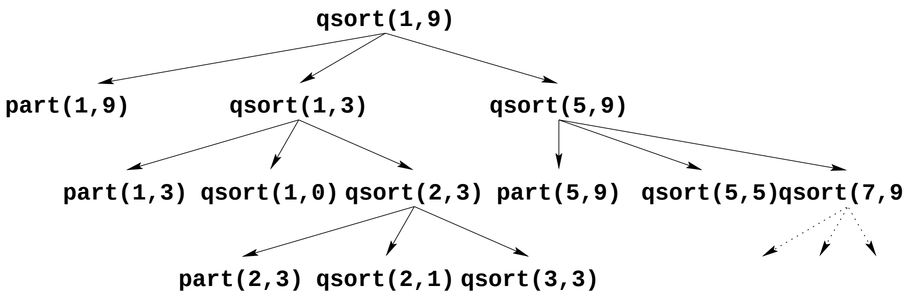
HEAP

Recursion

```
void qsort(int m, int n)
{
  int i;

  if (n > m) {
    i = part(m, n);
    qsort(m, i-1);
    qsort(i+1, n);
  }
}
```

Activation Trees

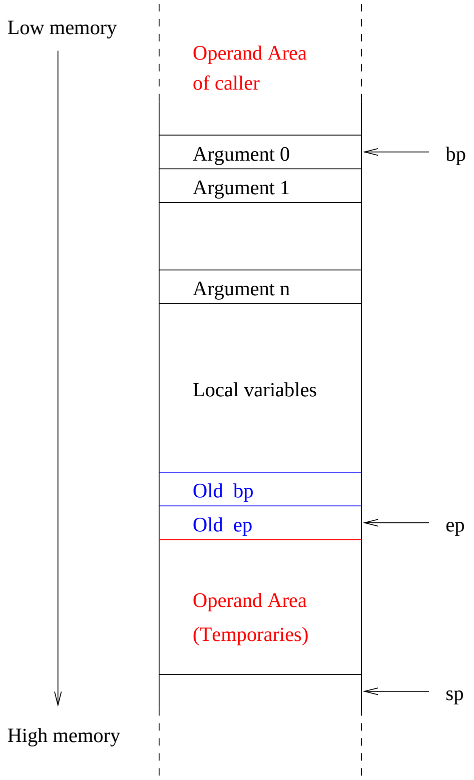


Activation Records

All information local to a *single* invocation of a procedure is kept in an *Activation Record*.

- Return Address
- Arguments
- Return Value
- Local variables
- Temporaries
- Other control information

Activation Records: An Example



Low memory

Operand Area
of caller

Argument 0

bp

Argument 1

Argument n

Local variables

Old bp

Old ep

ep

Operand Area
(Temporaries)

sp

High memory

Organizing Activation Records

Control information for accessing different areas in an activation record:

- **Base Pointer:** Beginning of activation record.
  Arguments are accessed as offsets from base pointer.
- **Environment Pointer:** Pointer to the most recent activation record.
  Usually a fixed offset from base pointer.
- **Stack Pointer:** Top of activation record stack.
  Temporaries are allocated on top of stack.

Managing Activation Records

```
int m(int k)
{
    int i;

    i = k + 15 * n(3);
    return l(i);
}
```

Managing Activation Records (contd.)

```
_m:
        pushl %ebp
        movl %esp,%ebp

        .. code for m

        movl %ebp, %esp
        popl %ebp
        ret
```