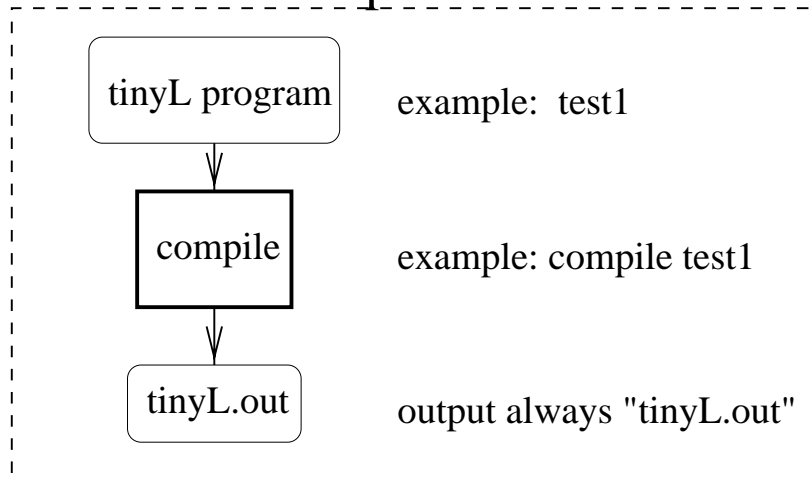


Review

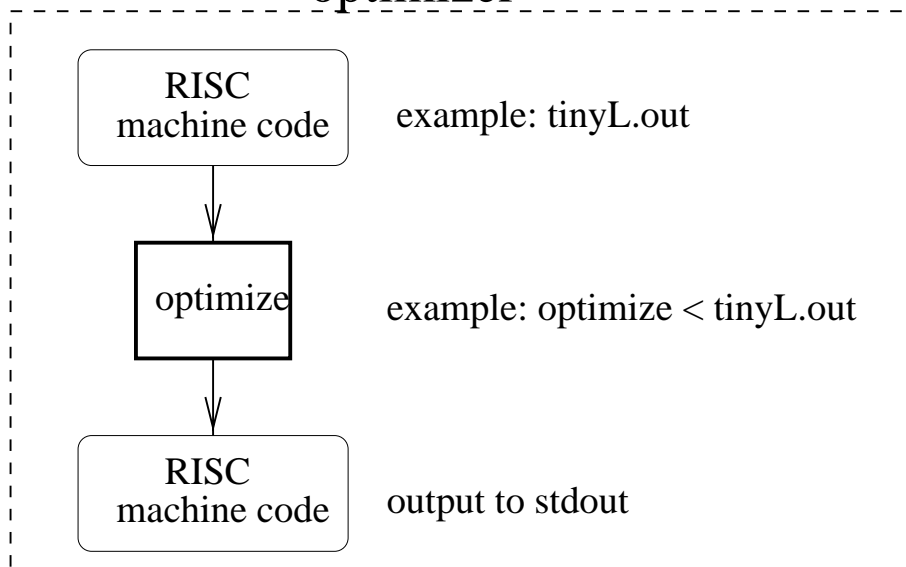
- how to compile, debug, and run C programs
- I/O in C (printf, scanf)
- pointers in C

Project: Overview

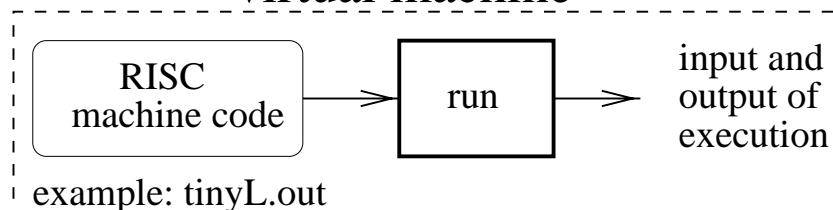
compiler



optimizer



virtual machine



Project: Makefile

```
compile: Compiler.c Parser.c CodeGen.c InstrUtils.c InstrUtils.h Ast.h CodeGen.h  
gcc Compiler.c Parser.c CodeGen.c InstrUtils.c -o compile
```

```
optimize: Optimizer.c InstrUtils.c InstrUtils.h CodeGen.c CodeGen.h  
gcc Optimizer.c CodeGen.c InstrUtils.c -o optimize
```

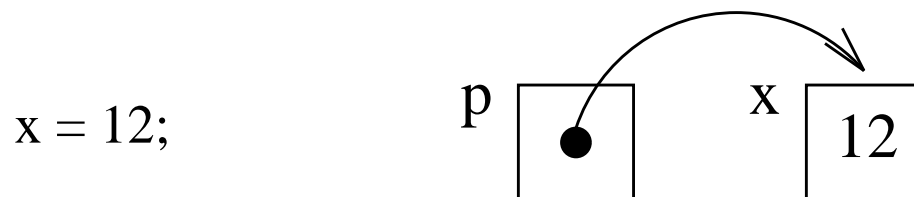
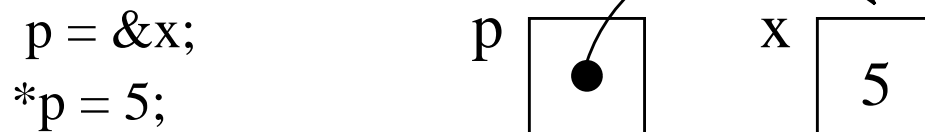
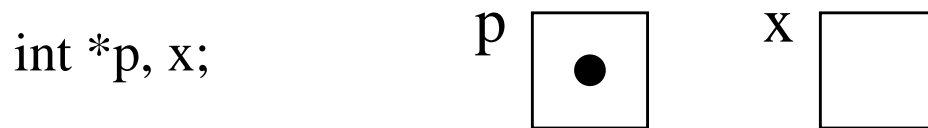
```
run: Interpreter.c InstrUtils.c InstrUtils.h CodeGen.c CodeGen.h  
gcc Interpreter.c CodeGen.c InstrUtils.c -o run
```

- You don't have to use a makefile!
- Keeps track of dependences between source code files.
- Automatically recompiles if at least one source file has more recent time stamp than the executable.
- Need to edit makefile if you want to debug your executable (insert `-g` option). Alternative: don't use makefile, and just enter command on Unix prompt.
- Very powerful tool (see `man make`).

Pointers in C

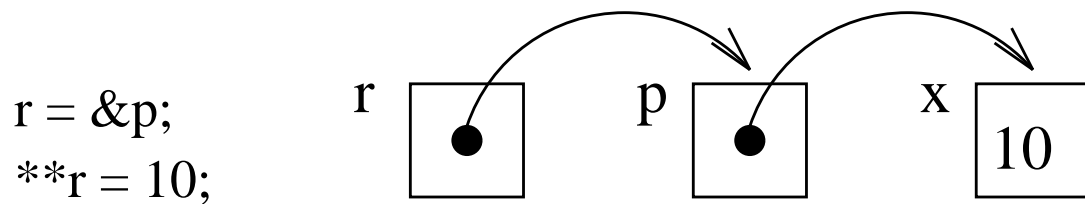
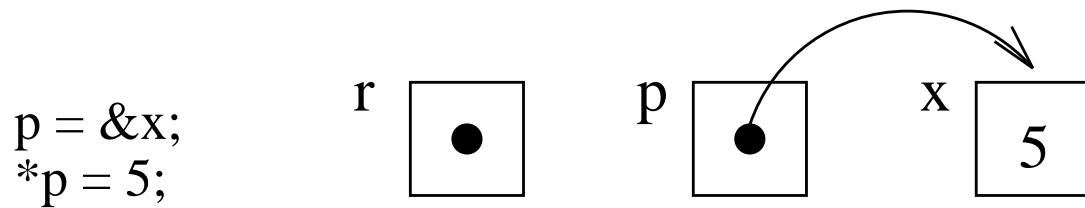
Pointer: Variable whose R-values (content) is the L-value (address) of a variable

- “address-of” operator `&`
- dereference (“content-of”) operator `*`



Pointers in C

- Pointers can point to pointer variables (multi-level pointers)

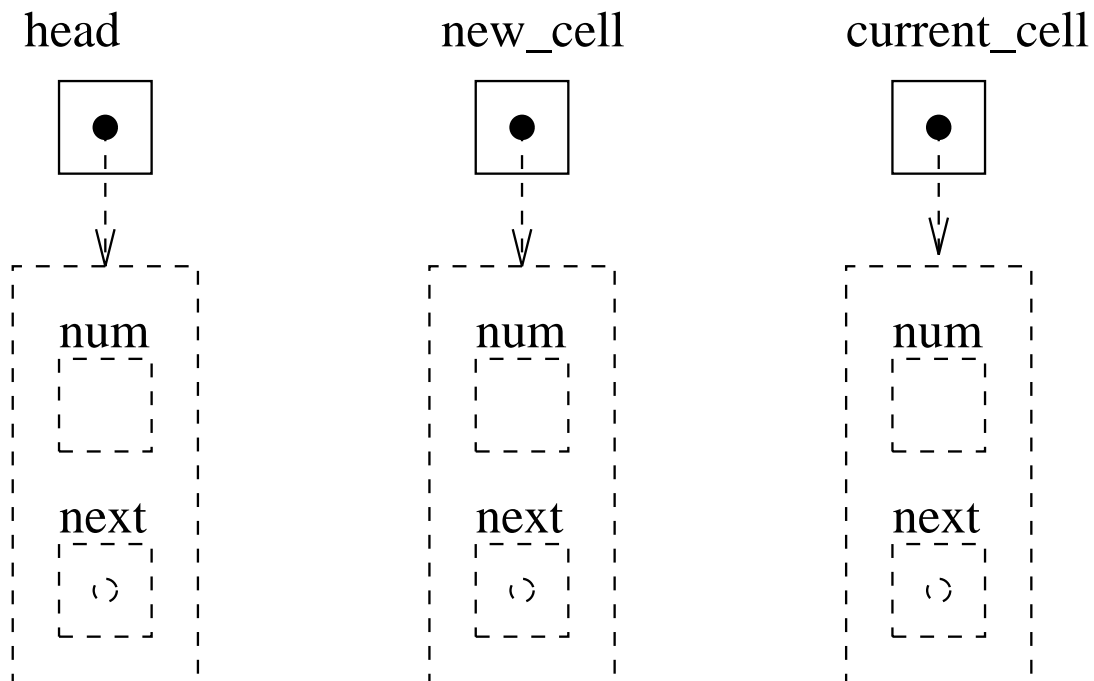


Example: Singly-linked list

list.h :

```
#ifndef LIST_H
#define LIST_H
#include <stdio.h>

/* TYPE DEFINITION */
typedef struct cell listcell;
struct cell
{ int num;
  listcell *next;
};
#endif
```



```
/* GLOBAL VARIABLES */
listcell *head, *new_cell, *current_cell;
```

Example: Singly-linked list

list.c :

```
#include "list.h"
/* GLOBAL VARIABLES */
listcell *head, *new_cell, *current_cell;
int main (void) {
    int j;

    /* CREATE FIRST LIST ELEMENT */
    head = (listcell *) malloc(sizeof(listcell));
    head->num = 1;      /* same as *h.num */
    head->next = NULL; /* same as *h.next */

    /* CREATE 9 MORE ELEMENTS */
    for (j=2; j<=10; j++) {
        new_cell = (listcell *) malloc(sizeof(listcell));
        new_cell->num = j;
        new_cell->next = head;
        head = new_cell;
    }

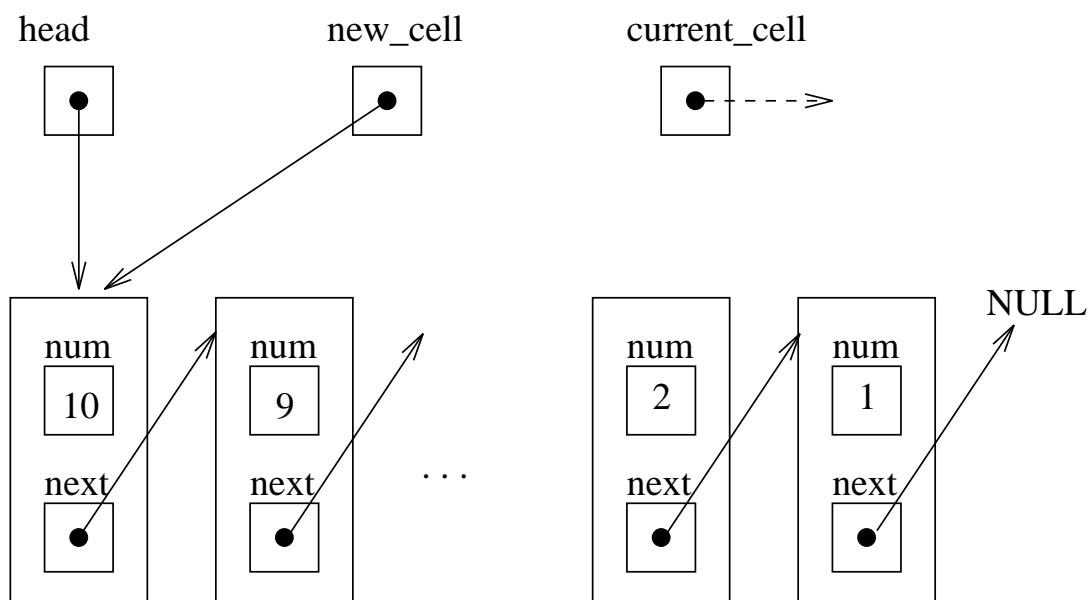
    /* PRINT ALL ELEMENTS */
    for (current_cell = head;
        current_cell != NULL;
        current_cell = current_cell->next)
        printf("%d ", current_cell->num);

    printf("\n");
    return 0;
}
```

Example: Singly-linked list

```
#include "list.h"
/* GLOBAL VARIABLES */
listcell *head, *new_cell, *current_cell;
int main (void) {
    int j;
    /* CREATE FIRST LIST ELEMENT */
    head = (listcell *) malloc(sizeof(listcell));
    head->num = 1;
    head->next = NULL;

    /* CREATE 9 MORE ELEMENTS */
    for (j=2; j<=10; j++) {
        new_cell = (listcell *) malloc(sizeof(listcell));
        new_cell->num = j;
        new_cell->next = head;
        head = new_cell;
    }
    /* *** HERE *** */
}
```



Example: Singly-linked list

- What is the output of the program
- Where do the cell objects get allocated?

Review: Stack vs. Heap

Stack:

- Procedure activations, statically allocated local variables, parameter values
- Lifetime same as subroutine in which variables are declared
- Stack frame is pushed with each invocation of a subroutine, and popped after subroutine exit

Heap:

- Dynamically allocated data structures, whose size may not be known in advance
- Lifetime extends beyond subroutine in which they are created'
- Must be explicitly freed or garbage collected

Heap Storage

```
void * malloc(size_t n)
```

- returns pointer to block of contiguous storage of **n** bytes on the heap, if possible
- returns NULL pointer if not enough memory is available
 - ⇒ you should check for `==NULL` after each `malloc`
- NOTE: we didn't do this in the example!
- to allocate storage of a desired type, call `malloc` with the needed size in bytes, and then cast the return pointer to the desired type

```
head = (listcell *) malloc(sizeof(listcell));
```

```
void free(void *ptr)
```

- data structure that `ptr` points to is released, i.e., returned to the free memory and may be (partially) reused by a subsequent `malloc`.

Next Lecture

Things to do:

Start working on project as soon as possible. Project has been posted.

Next time:

- more on pointers
- garbage collection