Class Information

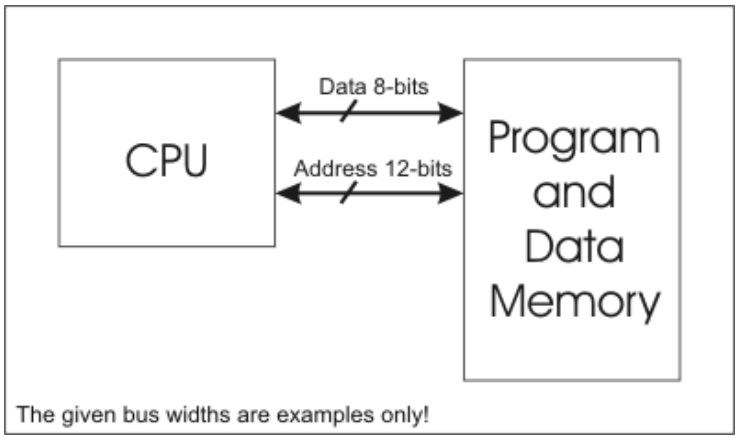
- Last chance to turn in your second homework.
- There are some general intermittened networking problems with the CS web sites. This is not specific to our 314 web site. If you don't have access, try later.
- The first programming project will be posted soon, tentatively tomorrow (Wednesday). Due on Friday, March 7.

Imperative Programming Languages

Imperative:

Sequence of state-changing actions.

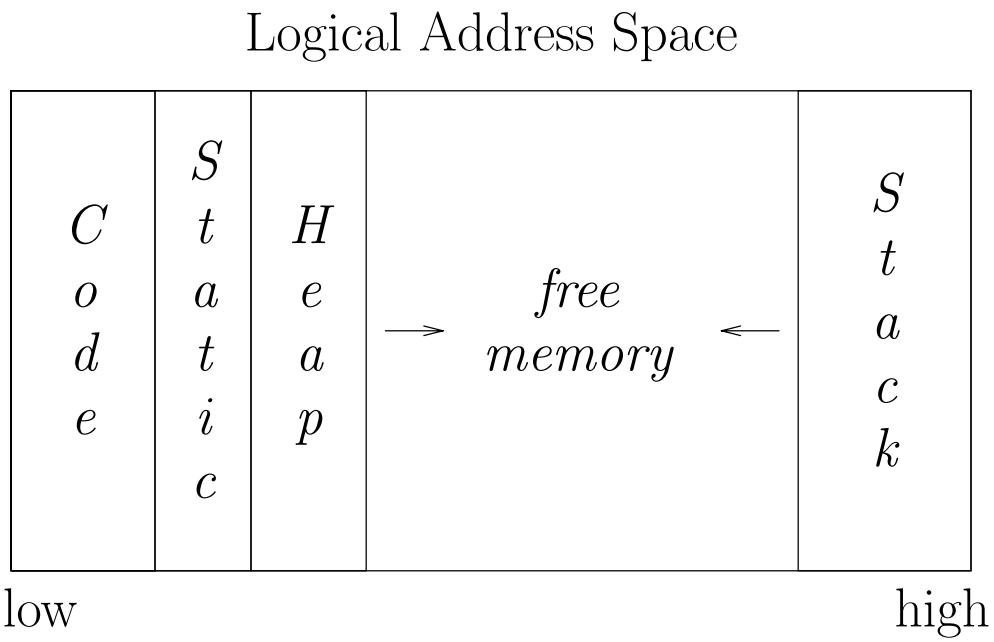
- Manipulate an abstract machine with:
  1. Variables naming memory locations
  2. Arithmetic and logical operations
  3. Reference, evaluate, assign operations
  4. Explicit control flow statements
- Key operations: *Assignment* and "*Goto*"
- Fits the von Neumann architecture closely



Von Neumann Architecture

Run-time storage organization

Typical memory layout



The classical scheme

- allows both stack and heap maximal freedom
- code and static may be separate or intermingled

Will talk about this in more detail in a later lecture!

C: An Imperative Programming Language

Expressions: include procedure and function calls and assignments, and thus can have side-effects

Control Structures:

- if statements, with and without else clauses
- loops, with break and continue exits

  `while` (<expr>) <stmt>
  `do` <stmt> `while` (<expr>)
  `for` (<expr> ; <expr> ; <expr>) <stmt>

- switch statements
- goto with labelled branch targets

C Examples

```
while ( ( c = getchar()) != EOF ) putchar(c);
```

```
for ( i = 0 ; s[i] == ' ' ; i++ );
```

```
for ( i = 0 ; i < n ; i++ ) {
  if ( a[i] < 0 ) continue; /*skip neg elems*/
  ....
}
```

```
c = getchar();
switch(c) {
     case '0': case '1': case '2': case '3':
     case '4': case '5': case '6': case '7':
     case '8': case '9':
        digit[c-'0']++;
        break;
     case ' ': case '\n': case '\t':
        delim++;
        break;
     ...
 }
```

Data Types in C

- Primitives: `char`, `int`, `float`, `double`
  no Boolean—any nonzero value is true
- Aggregates: arrays, structures

```
char a[10], b[2][10];

struct rectangle {
   struct point p1;
   struct point p2;
}
```

- Enumerations: collection of sequenced values
- Pointers:

`&i`	address of i
`*p`	dereferenced value of p
`p+1`	pointer arithmetic

```
int *p, i;
p = &i;
*p = *p + 1;
```

Basic Comparison (incomplete!)

C	Java
Basic types: int, double, char	Primitive types: int, double, char, **boolean**
Pointer (to a value)	**Reference (to an object)**
Aggregates: array, **struct**	Aggregates: array, object (**class**)
Control flow: if-else, switch, while, break, continue, for, return, **goto**	Control flow if-else, switch, while, break, continue, for, return
Logic operators: ||, &&, !	Logic operators: ||, &&, !
Logical comparisons: ==, !=	Logical comparisons: ==, !=
Numeric comparisons: <>, <=, >=	Numeric comparisons: <>, <=, >=
string as **char * array**	**String** as an object

Compile and Run a C program

`test.c`:

```
#include <stdio.h>

int
main(void)
{
  int x, y;

  printf("First number:\n"); scanf("%d", &x);
  printf("Second number:\n"); scanf("%d", &y);

  printf("%d+%d = %d\n", x, y, x+y);
  printf("%d-%d = %d\n", x, y, x-y);
  printf("%d*%d = %d\n", x, y, x*y);

  return 0;
}
```

`gcc test.c`:	calls the GNU C compiler, and generates executable `a.out`
`./a.out`	runs the executable
`gcc -o run test.c`	compiles program, and generates executable `run`
`gcc -g test.c`	generates `a.out` with debugging info
`gdb a.out`	run debugger on `a.out`; online documentation `man gdb`

Compile and Run a C program

```
> gcc test.c
> a.out
First number:
4
Second number:
12
4+12 = 16
4-12 = -8
4*12 = 48
>
```

START PROGRAMMING IN C NOW!

Debugging C programs

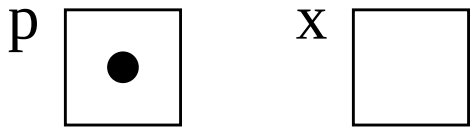
```
rhea% gdb a.out
(gdb) list
1       #include <stdio.h>
2       int main(void)
3       {
4         int x, y;
5         printf("First number:\n"); scanf("%d", &x);
6         printf("Second number:\n"); scanf("%d", &y);
7         printf("%d+%d = %d\n", x, y, x+y);
(gdb) break 7
Breakpoint 1 at 0x1052c: file test.c, line 7.
(gdb) run
Starting program: /.../a.out
First number:
4
Second number:
12
Breakpoint 1, main () at test.c:7
7         printf("%d+%d = %d\n", x, y, x+y);
(gdb) print x
$1 = 4
(gdb) print y
$2 = 12
(gdb) cont
Continuing.
4+12 = 16
4-12 = -8
4*12 = 48
Program exited normally.
(gdb) quit
```

Pointers in C

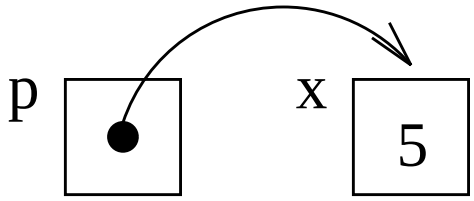
Pointer: Variable whose R-values (content) is the L-value (address) of a variable

- "address-of" operator &
- dereference ("content-of") operator $*$

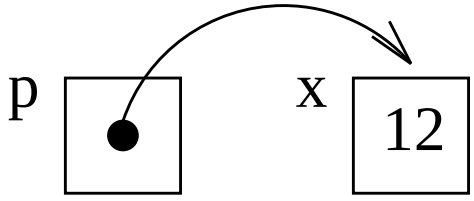
```
int *p, x;
```




```
p = &x;
*p = 5;
```



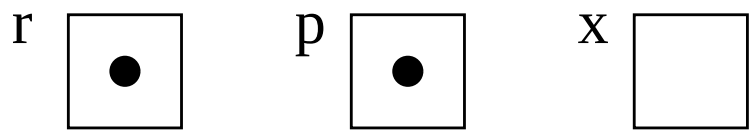

```
x = 12;
```



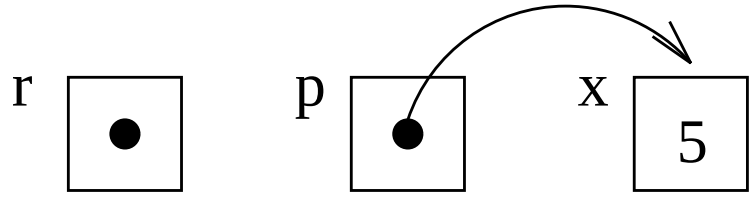
Pointers in C

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

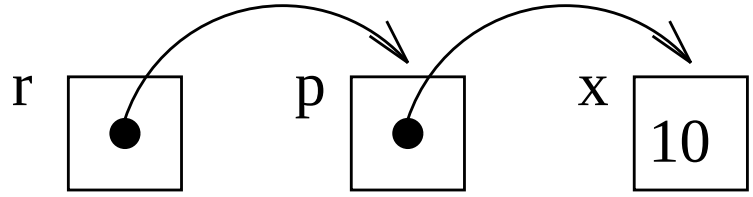
```
int *p, x;
int **r;
```




```
p = &x;
*p = 5;
```




```
r = &p;
**r = 10;
```

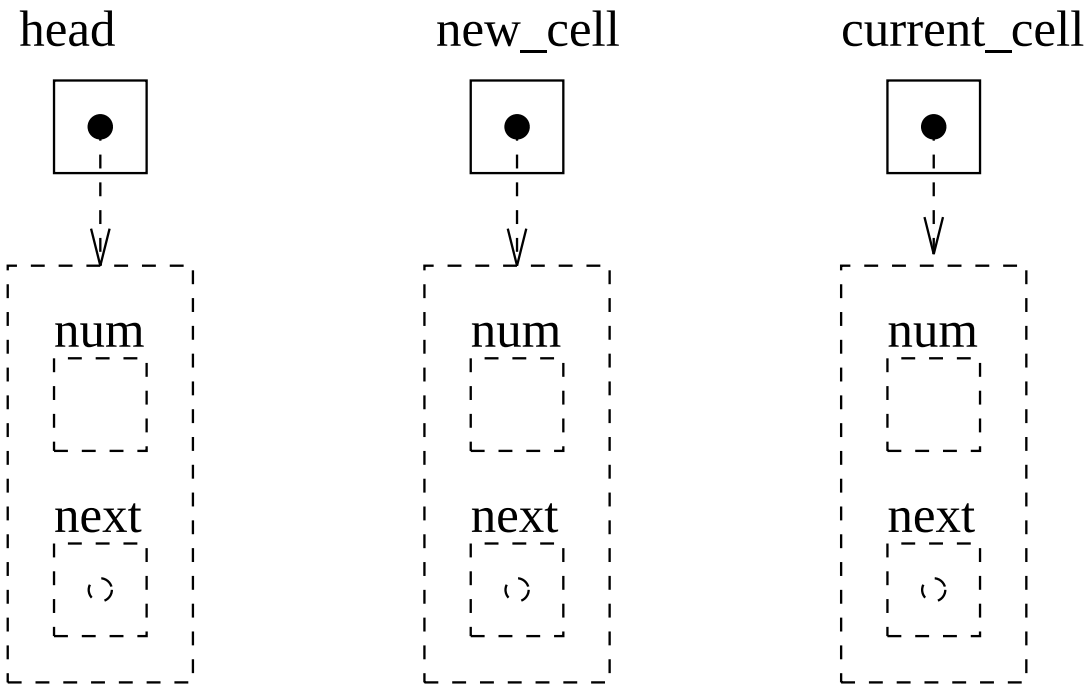


Pointers in C vs. References in Java

C	Java
Need explicit dereference operator $*$	are implicitly dereferenced
Can mutate R-value of pointer through pointer arithmetic `p=p+1`	Cannot mutate R-value
Casting means type conversion	Casting just satisfies the type checker; no type conversion
Special relation to arrays	No special relation to arrays

Example: Singly-linked list

```
#include <stdio.h>
#include <stdlib.h>
/* TYPE DEFINITION */
typedef struct cell listcell;
struct cell
{ int num;
  listcell *next;
};

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

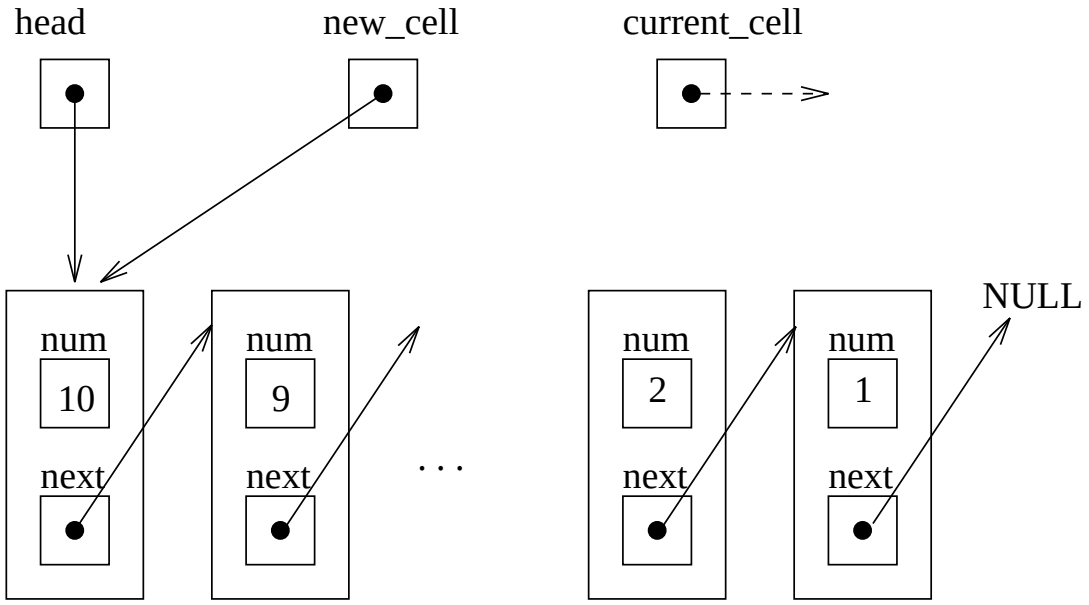


Example: Singly-linked list

```
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;
  }

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

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

Example: Singly-linked list

```
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 poped 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

Next Lecture

Things to do:

Start programming in C. Check out the web for tutorials.

Read Scott: Chap. 8.1 - 8.2 ; ALSU Chap. 7.1 - 7.3

Next time:

- More about programming in C.
- Procedure abstractions; run time stack; scoping.