# Imperative Languages: Data and Assignments



CS314 Spring 2001© L Steinberg, B. Ryder

# RAM: Random Access Machine

- **Normal control flows from one instruction to the next**
  - ***Thread of computation*: sequence of program points reached as execution flows through the program**
- **Control flow directs the thread without changing state**
- **Data flow (through assignment) affects the state without directly affecting the thread**
- **Imperative PLs have primitives close to the machine instructions (e.g., assignment, branch)**



CS314 Spring 2001© L Steinberg, B. Ryder

# RAM

- A universal computing device, similar to a Turing Machine
- Components:
  - Program-sequence of instructions
  - Memory-sequence of locations
  - Control-current location in program
  - Input file- sequence of values
  - Output file-sequence of values
- Control flows from one instructions to next

# RAM Instructions

- **Assignment:**

  **M[j]=k;**

  **M[j]=M[l]+M[n];**

  **M[j]=M[l]-M[n];**

  **M[j]=M[M[l]];**

  **M[M[j]]=M[k];**
- **Input: read M[k];**
- **Output: write M[k];**
- **Branch: if M[k] >= 0 then goto loop**
- **Halt: halt;**



CS314 Spring 2001© L Steinberg, B. Ryder

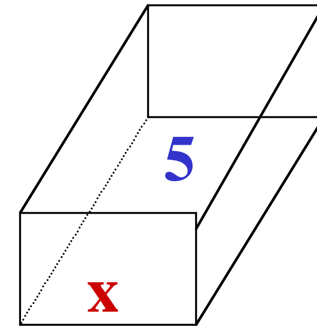
# L-values and R-values

- **L-value of location M[j] is location j**
- **R-value of location M[j] is contents of location j**
  - **M[j] = M[k] means fetch the r-value of location k and store it into location j or, alternatively, store the r-value of M[k] into the l-value of M[j]**
  - **Indirect addressing**
    - **L-value of M[M[j]] is location M[j];**
    - **R-value of M[M[j]] is contents of location M[j]**


CS314 Spring 2001© L Steinberg, B. Ryder

# C, An Imperative PL

- Assignment as main operation
  - Names ⇔ Locations ⇔ Values
  - L-value: name labeling memory location
  - R-value: contents of memory location




- State of a computation
  - M: Locations -> Values
  - Remaining input
  - Output so far



CS314 Spring 2001© L Steinberg, B. Ryder

# Bird's Eye View: C versus *Java*

- **Types: int, double, char**
- **Pointer (to a value)**
- **Aggregates: array, struct**
- **Control flow: if-else, switch, while, break, continue, for, return, goto**
- **Logic operators: || && !**
- **Logical comparisons: == !=**
- **Numeric comparisons: < > <= >=**
- **string as char * array**

- ***Primitive types: int, double, char, boolean***
- ***Reference (to objects)***
- ***Aggregates: array, object***
- ***Control flow: if-else, switch, while, break, continue, for, return***
- ***Logic operators: || && !***
- ***Logical comparisons: == !=***
- ***Numeric comparisons: < > <= >=***
- ***String* as an object**

# C Datatypes

Sethi, p 583ff

- **Primitive: char, int, double (+others)**
  - **Without boolean type, any nonzero value is *true*; zero is *false***
- **Aggregates: arrays, structs**
  - **homogeneous arrays**

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

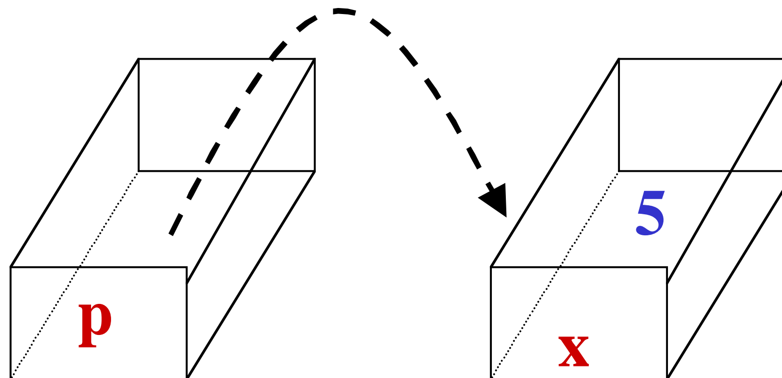
  - **heterogeneous structs**

```
struct employee{
  int age;
  double payrate;
}

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

# C Datatypes

- **Pointers: variables whose value is the L-value of a variable**
  - **address-of operator &**
  - **dereference operator for a pointer to obtain its R-value ***

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





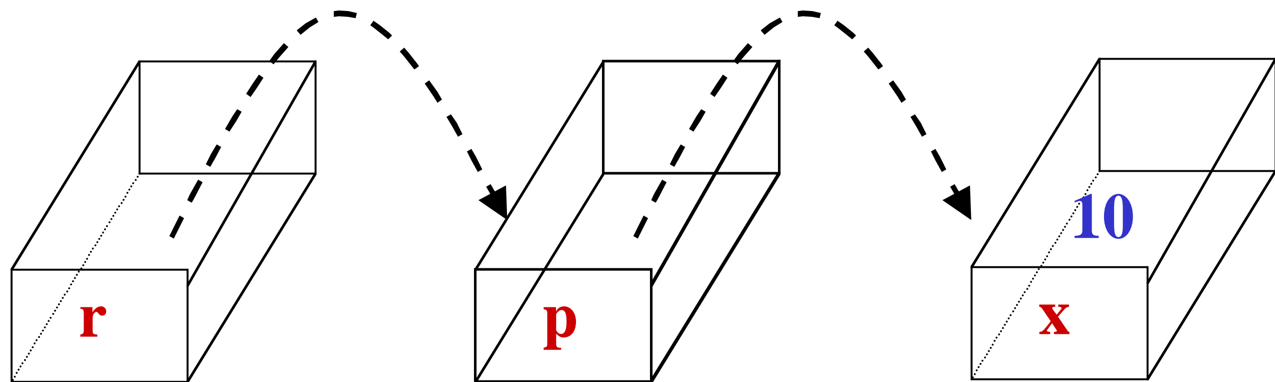


CS314 Spring 2001© L Steinberg, B. Ryder

# C Datatypes

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

```
int *p;
int ** r;
p = &x;
*p = 5;
r = &p;
**r = 10;
```






CS314 Spring 2001© L Steinberg, B. Ryder

# Pointers versus References

- Need explicit dereference operator *
- Can mutate R-value of a pointer through pointer arithmetic

  p = p +3;

- Casting means type conversion of kind of value pointed to
- Special relation to arrays

- Are implicitly dereferenced
- Cannot mutate R-value of a reference
- Casting just satisfies the type checker; does no type conversion
- No special relation to arrays



CS314 Spring 2001© L Steinberg, B. Ryder

# Pointers and Arrays

- **An array name is considered pointer to first element**
  - **a is pointer to a[0]**
  - **pa = &a[0] and pa=a mean the same thing**
  - **a+1 means L-value of a[0] plus as many bytes as are needed to store value of elements of a's type**
    - **Pointer arithmetic is an address calculation with respect to the underlying architecture**
- **An array name is not a variable**
  - **a++ and a = pa are ILLEGAL!**



CS314 Spring 2001© L Steinberg, B. Ryder

# C Example, hello.c

```
/*sample program to do hello world
  and print the numbers from 1 to 10*/
#include<stdio.h>
main (void)
{    int j, n;

     printf(" hello world\n");
     n = 10;
     for (j = 0; j <= n; ++j)
       printf(" %d",j);
     printf("\n the even numbers are: ");
     for (j = 0; j < 11; ++j)
       if ((j%2) == 0) printf(" %d",j);

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

# C Example Output

```
/*when hello.c is run it looks like
  this:
scherzo!c> gcc hello.c
scherzo!c> a.out
 hello world
 0 1 2 3 4 5 6 7 8 9 10
 the even numbers are:  0 2 4 6 8
  10
scherzo!c>

*/
```



CS314 Spring 2001© L Steinberg, B. Ryder

# Multipleptr.c

```
#include<stdio.h>
/*program to show multiple levels of dereference*/
main( )
{        int j, k, l;
         int *q;
         int **s;
         j = 99;
         q = &j; /* q = j is ILLEGAL */
         s = &q; /* s = q is ILLEGAL */
/*all these names **s, *q, j are synonyms or aliases
  for the same storage location at this program point*/
         printf(" %d %d %d\n",**s, *q, j);
}
/* output
47 scherzo!c> gcc multipleptr.c
48 scherzo!c> a.out
 99 99 99  */
```



CS314 Spring 2001© L Steinberg, B. Ryder

# Problems with Pointers

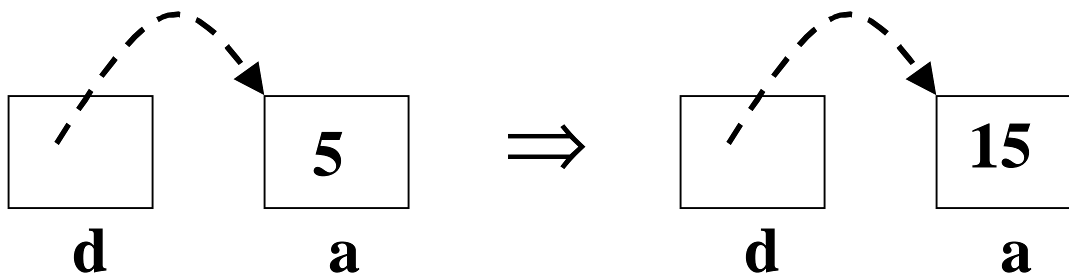
- **Uninitialized pointers**
  - **int *a;  *a = 12;**
    - **Can get *segmentation fault* error (value in a is meaningless address)**
    - **Can actually store 12 into some memory location accessible to your program, whose address corresponds to the random bits in a**



CS314 Spring 2001© L Steinberg, B. Ryder

# Problems with Pointers

- **Null** doesn't point to anything by definition so cannot dereference it
  - a = 0; /* makes a's value NULL */
  - if (a == NULL) … /* tests for a NULL pointer value*/
- Multiple level pointers
  - Can be used as L-values or R-values




```
int a;
int *d;
d = &a;
a = 5;
*d = 10 +*d;
a  =  10 + 5
```

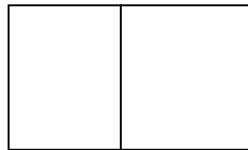


CS314 Spring 2001© L Steinberg, B. Ryder

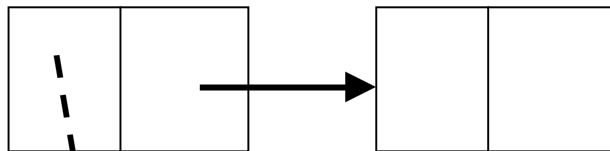
# Exerpt from List in Java

```
public class List extends Object {
protected Object element;
protected List subList;
  /**
   * Create an new List, initially empty.
   */
  public List() {
   element = null;
   subList = null;
  }

//cons operation
  public List(Object newElement,List oldList){
   element = newElement;
   subList = oldList;
} }
```

# How Java List works

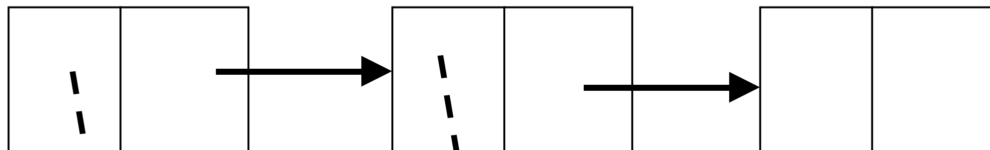


List p = new List();



List q = new List(“a”,p);

“a”



List(“b”,q);

“b”

“a”

# list.c

```
/*sample program to write a linear linked list of
  integers built like in Java, adding new elements on
  the front*/
#include<stdio.h>
/*this makes these definitions and variables globals*/
typedef struct cell listcell;
struct cell{
  int num;
  listcell *next;
};
listcell *head, *ele, *p;
```

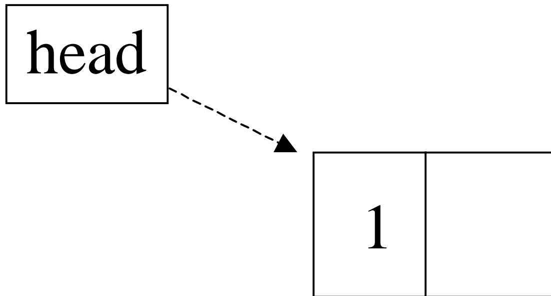
# list.c, cont.

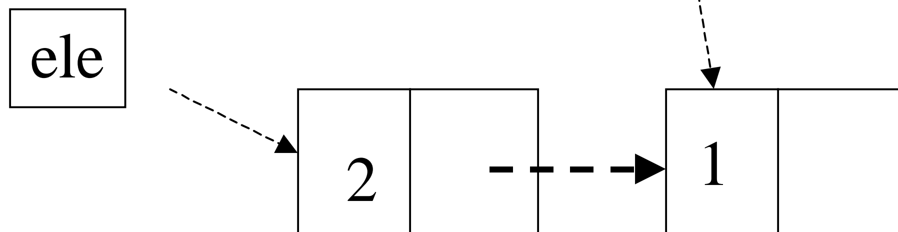
```
main(void)
{  int j;
/*create first node in list*/
  head = (listcell *) malloc(sizeof (listcell));
  head->next = NULL;
/*now create other entries in list of numbers from 1
  to 10*/
  head->num = 1;
  for (j=2; j<11; j++)
  {   ele = (listcell *) malloc(sizeof (listcell));
      ele->num = j;
      ele->next =head; /***/
      head = ele;
  }
/*now traverse the list and print the elements*/
  for (p=head; p!=NULL; p=p->next)
      printf("%d ",p->num);
  printf("\n");
}
```


CS314 Spring 2001© L Steinberg, B. Ryder

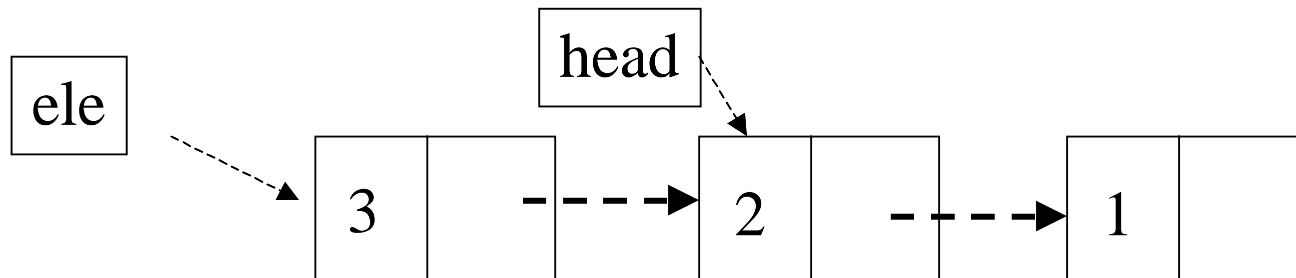
# list.c

*before first for loop.*




*at /***/ in 1st iteration*




*at /***/ in 2nd iteration*



CS314 Spring 2001© L Steinberg, B. Ryder

# list.c output

```
scherzo!c> gcc list.c
scherzo!c> ./a.out
10 9 8 7 6 5 4 3 2 1
scherzo!c>
```


CS314 Spring 2001© L Steinberg, B. Ryder

# *Stack* vs Heap

- *Procedure activations, statically allocated local variables, parameter values*
- *Lifetime same as block in which variables are declared*
- *Stack frame with each invocation of procedure*

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


CS314 Spring 2001© L Steinberg, B. Ryder

# Heap Storage

- ***void *malloc (size_t n)***
  - returns pointer to block of contiguous storage of *n* bytes (chars), if possible
  - if not enough memory left for allocation, *malloc* returns NULL pointer
    - So you ALWAYS have to check return the value
  - to allocate storage of a different type requires sending *malloc* the proper amount of bytes needed and casting the return pointer value appropriately

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

# listwithfree.c

```
/*sample program to write a linear linked list of
  integers built like in Java, adding new elements on
  the front*/
#include<stdio.h>
/*this makes these definitions and variables globals*/
typedef struct cell listcell;
struct cell{
  int num;
  listcell *next;
};
listcell *head, *ele, *p;
```

Same declarations as list.c



CS314 Spring 2001© L Steinberg, B. Ryder

# listwithfree.c, cont.

```
main(void)
{  int j;
/*create first node in list*/
   head = (listcell *) malloc(sizeof (listcell));
/*now create other entries in list of numbers from 1
   to 10*/
   head->num = 1;
   for (j=2; j<11; j++)
   {   ele = (listcell *) malloc(sizeof (listcell));
       ele->num = j;
       ele->next =head; /***/
       head = ele;
   }
/*now traverse the list and print the elements*/
   for (p=head; p!=NULL; p=p->next)
       printf("%d ",p->num);
   printf("\n");
}
```

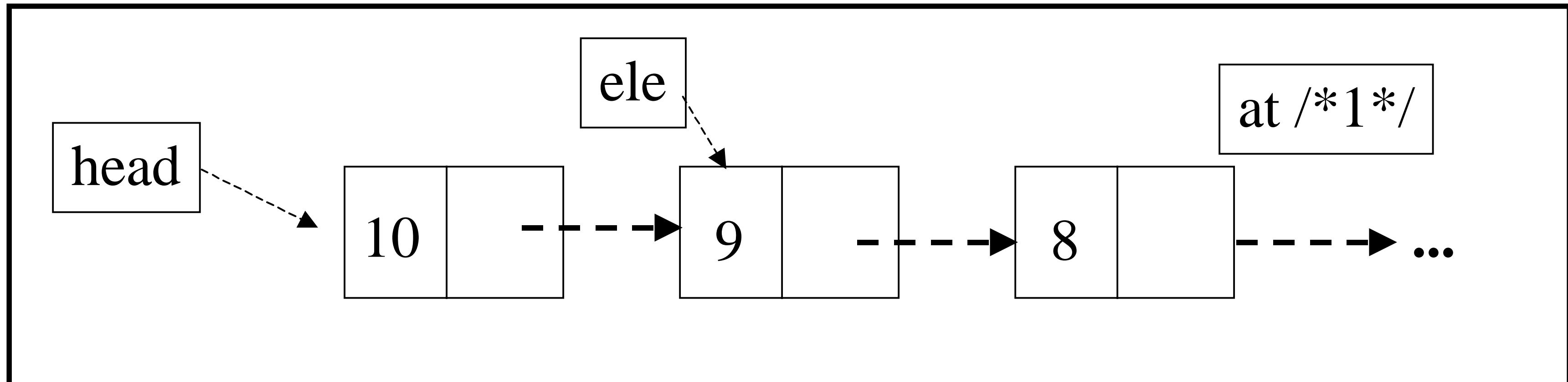
**Same building of the list and printing it out as list.c**

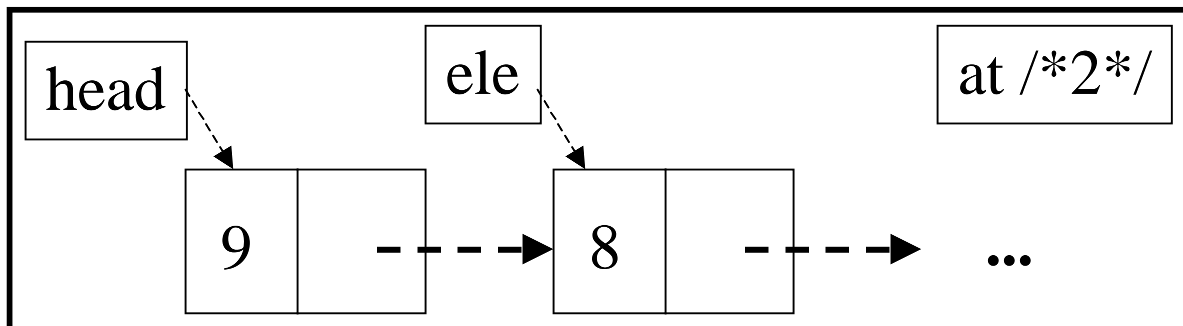
# listwithfree.c, cont.

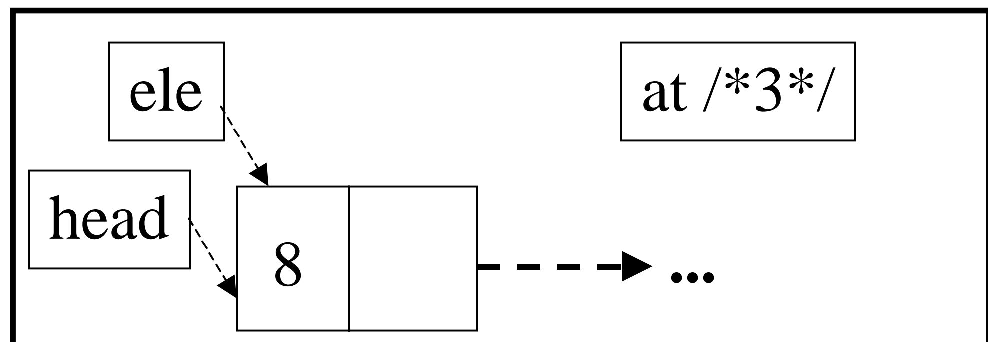
```
/*now delete the first 2 elements of the list and
  free their storage */
  ele = head->next;   /*1*/
  free (head); /* free 1st list element storage*/
  head = ele;
  ele = head->next; /*2*/
  free (head); /* free 2nd list element storage*/
  head = ele; /*3*/

/*now traverse the list and print the elements*/
  for (p=head; p!=NULL; p=p->next)
      printf("%d ",p->num);
  printf("\n");

}
```


CS314 Spring 2001© L Steinberg, B. Ryder

# Trace



# listwithfree.c, cont.

```
/* output
59 scherzo!c> a.out
10 9 8 7 6 5 4 3 2 1
8 7 6 5 4 3 2 1
60 scherzo!c>     */
```