INF-1100
Dynamic memory management and linked lists

Åge Kvalnes

University of Tromsø, Norway

October 30, 2014

Recap: Variables

- A variable has a type and a name.
  - The type of a variable limits the range of values that can be assigned to it.
- A variable is stored in memory at some address.
  - The type of the variable decides how much memory the variable occupies
  - To get the memory address of variable x, write &x.
- A variable declared outside a function remain in memory until the program terminates.
- A variable declared inside a function only remain in memory while instructions inside the function are executing.

Recap: Pointers

- A pointer is a variable containing **a memory address**.
- To declare a pointer variable specify type, followed by a $*$, and the name of the variable.
  - int *x; x is a pointer to an integer.
- A pointer variable can **only be assigned a memory address**.
  - int *x = &y; store the address of the variable y in the pointer variable x.
- The type of a pointer variable limits the **range of memory addresses** that can be assigned to it.
  - Must be the memory address of a variable whose type is the same as the pointer's type.
  - int *x = &y; is only ok if the type of y is int.
  - (Can use type casting to force assignment.)

Recap: Arrays

- An array is a variable containing **a sequence of variables of a specific type**.
- To declare an array specify the type of each variable, followed by the name of the array, followed by brackets specifying the size of the array.
  - int x[10]; x is an array of 10 integers.
- You can only assign or read array variables individually.
  - x[0] = 20; assign value 20 to array variable 0.
- The number of variables in an array is **fixed**.

Recap: Structs

- A struct is a **specification of a collection of variables of a specific type**.
  - struct {int x;}
- A struct is **not a variable, but a specification of a new data type**.
  - struct z {int x;}; z is the name of a struct containing an integer x.
  - struct z y; y is a variable, whose type is struct z.
- Structs are used for grouping together related variables so that the program becomes less 'messy'.
- Given a variable whose type is a struct, you use a dot notation to access the individual variables of the struct:
  - y.x = 20;
- Given a **pointer** to a variable whose type is struct, you use an array notation to access the individual variables of the struct:
  - struct z *p = &y;
  - p→x = 20;

Using arrays to store data

- A program often reads input data from an external source.
  - E.g., the program might read a number of variables from a file.
  - Or, the program might read text typed in by the user of the computer.
- A program often creates new variables depending on some condition
  - E.g., in a game the player shoots at some object on the screen. A variable is needed to describe the bullets (position, speed, etc.).
- Problem: An array has a fixed size. What if the number of variables needed exceeds the size of the array?

Dynamic memory allocation: malloc() and free()

- **malloc(n)** allocates **n** bytes of memory
  - Returns the starting address of n contiguous bytes of memory that are not in use by the program.
  - The return value is a memory address (i.e. a pointer).
- **free(p)** releases a chunk of memory.
  - **p** must be the return value from a previous malloc() call.
  - Do not use the memory after a call to free.
- Think of malloc() and free() as an interface to code that keeps track of memory not in use by the program.
  - The program calls malloc() to obtain a memory location for storing a new variable.
  - The program calls free() when a variable is not needed anymore.

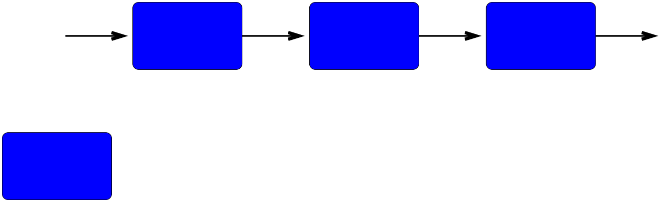
Linked lists

- Scenario: A program uses malloc() to allocate memory for many new variables.
  - The new variables are stored at different memory locations, i.e. wherever malloc() says there is free memory.
- Problem: How does the program keep track of the memory locations of the new variables?
  - Use an array of pointers to the variables? Need to know the number of variables then..No need to use malloc() if the number of variables is known in advance.
- Solution: Use linked lists
  - A way to organize variables into a logical sequence regardless of their memory location.
  - Place each variable in a *node*, and have each node contain the variable as well as the address of the next variable (node containing a variable).
  - Maintain a pointer to the first node in the list.

Linked lists

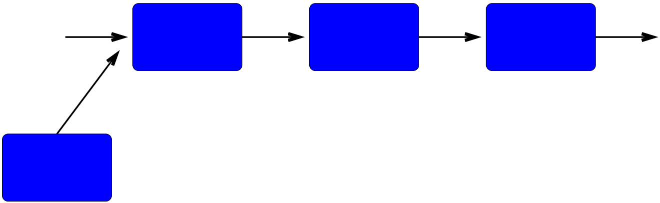
```
struct node {
   int x; // The variable
   struct node *next; //Pointer to next node
}

struct node *list;

list->x = 7;
list->next->x = 12;
list->next->next->x = 20;
```

Linked list insertion



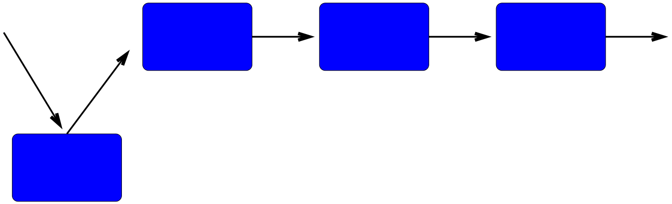
```
struct node {
   int x; // The variable
   struct node *next; //Pointer to next node
}

struct node *list;
struct node *new;

new = malloc(sizeof(struct node));
```

Linked list insertion



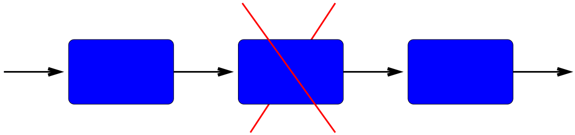
```
struct node {
   int x; // The variable
   struct node *next; //Pointer to next node
}

struct node *list;
struct node *new;

new = malloc(sizeof(struct node));
new→next = list;
```

Linked list insertion

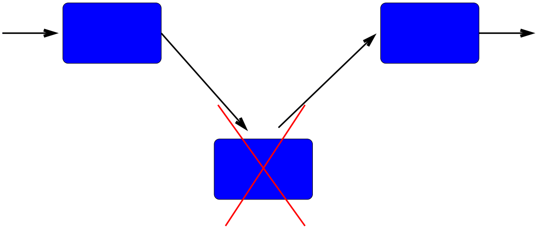


```
struct node {
   int x; // The variable
   struct node *next; //Pointer to next node
}

struct node *list;
struct node *new;

new = malloc(sizeof(struct node));
new→next = list;
list = new;
```

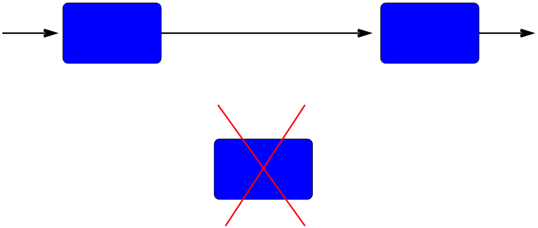
Linked list removal



Linked list removal



Linked list removal



Linked list traversal

```
struct node {
   int x; // The variable
   struct node *next; //Pointer to next node
}

struct node *list;
struct node *current;

current = list;
while (current != NULL) {
   do something with the variables in the node;
   current = current→next;
}
```

ClickTracker: tracking use of search results

Google receives a large fraction of its income from advertisement.

- Advertisements are placed on search result pages.
- Google is paid whenever you click on an advertisement.

Clickthrough.

- Click goes to Google then to the real destination.
- Enables Google to record your clicks.

ClickTracker: cookies

Cookies are data strings stored on your computer.

- Supplied by a Web site and stored by the Web browser.
- Presented to the Web site when it is re-visited.

Cookies can be used to identify you.

- Personalization of Web content.
- Automatic login.

Used by Google to keep track of how you use their search results.

ClickTracker

Problem: Store information about what URIs a particular user clicks on in the search result list.

- Users are uniquely identified by a cookie.
- What URI the user clicks on is known.

Interface:

- **void ClickRegister(char *uri, char *cookie)**
- **int ClickNumURI(char *uri)**
- **int ClickNumCookie(char *cookie)**