# C PROGRAMMING II

Ruben Acuña

Calvin Cheng

Spring 2019

## 3 DEFINING NEW TYPES

## CREATING TYPES WITH TYPEDEF

The *typedef* keyword introduces a new name that becomes a synonym for the type given by the typename portion of the declaration.   For example:

Wait, why not just use char directly?

```
//example 1
typedef int boolean;

void main() {
    boolean x = 0;
    int counter = 5;
    if (x == 0) counter++;
}
```

```
//example 2
typedef char FlagType;

int main() {
    FlagType x = '\0';
    ...
}
```

## DECLARING ENUMERATIONS

Another keyword is *enum*, which allows us to define a type that is to be populated by specific elements (symbols). In C, each enum value is an actual int, and can be directly manipulated.  Note that an enum value is not type safe.

```
//Example 1:
enum { false, true } boolean;
void main() {
     enum boolean x = false;
     int counter = 5;

     if (x == true) counter++;
}

//Example 2:
enum {
    Sun=0, Mon, Tue, Wed, Thu, Fri, Sat
} days;
enum days day = Mon;
```

Forces enum elements to be numbered from zero – otherwise their values are not well defined.

## USING ENUMERATIONS

Recall in that the previous example we created a variable with "enum days day;". This is a little awkward because we have to write "enum days" for the type name. We can fix this with typedef though:

```
enum { Sun=0, Mon, Tue, Wed, Thu, Fri, Sat } days;
typedef enum days Days;
Days now = Mon;
```

As a final thought about type safety, consider the following code.

- What is the final value of x?
- What is the value of x, after x++ is executed and if x was Sat before?

```
Days x = Mon, y = Fri;
while (x != y)
    x++;
```

## COMPOUND DATA TYPE: STRUCT

A structure is created by the keyword *struct.* A structure is composite data type, similar to a class in Java but only containing state.

```
struct type_name {
       type1   element1;
       type2   element2;
       …
       typen   element;
};
struct type_name a, b;
a.element1 = 10;
```

Don't like having to put struct in front of the typename? Use typedef.

- Why is structure useful?

## STRUCT EXAMPLE

```c
#include <stdio.h>
#include <string.h>

struct Course {
    int sln;
    char title[256];
    char instructor[250];
    int time;
};

struct Course ser222;

void main() {
     ser222.sln = 1774;
     strcpy(ser222.title, “Data Structures & Algorithms");
     strcat(ser222.instructor, “Acuna, Ruben");

     printf("Memory allocated for this structure is %d", sizeof(ser222));
}
```

How much memory is allocated for this structure?

## STRUCTURE PADDING

```
struct Course {
    int sln;
    char title[256];
    char instructor[250];
    int time;
} ser222;
```

| sln | title | instructor | | time |
|---|---|---|---|---|
| 1774 | Data Structures & Algorithms | Acuna, Ruben | 🟥 | 0 |
| 4-bytes | 256-bytes | 250-bytes | 2-bytes (padding) | 4-bytes |

Padding is needed to align the data in memory; in such way, a computer processor can read 1 word from memory at a time. (In a 32-bit processor it is 4 bytes and 8-bytes for a 64-bit processor.)

It is important (why?) to align the data correctly to avoid unnecessary paddings.

## ACCESSING MEMBERS OF A STRUCTURE

```
#include <stdio.h>
#include <string.h>

struct Course {
    int sln;
    char title[256];
    char instructor[250];
    int time;
};

struct Course ser222;
struct Course* cse360;

void main() {
      // Direct access to the data (stack memory)
      ser222.sln = 1774;
      strcpy(ser222.title, “Data Structures & Algorithms");
      strcpy(ser222.instructor, “Acuna, Ruben");

      // Dynamic Allocation in heap memory
      cse360 = (struct Course*)malloc(sizeof(struct Course));

      // Indirect access (using pointer) to the data (heap memory)
      (*cse360).sln = 2234; //same as arrow notation below
      strcpy(cse360->title, "Introduction to SW Engineering");
      strcpy(cse360->instructor, "Tsai, Wei-Tek");
}
```

## COMPOUND DATA TYPE: UNION

- A *union* is a region of shared memory that, over time, can contain values with different data types.
- At any moment, a union can contain only one value. (Any way to make this easy?) Programmers must make sure the proper type is used at the proper time.

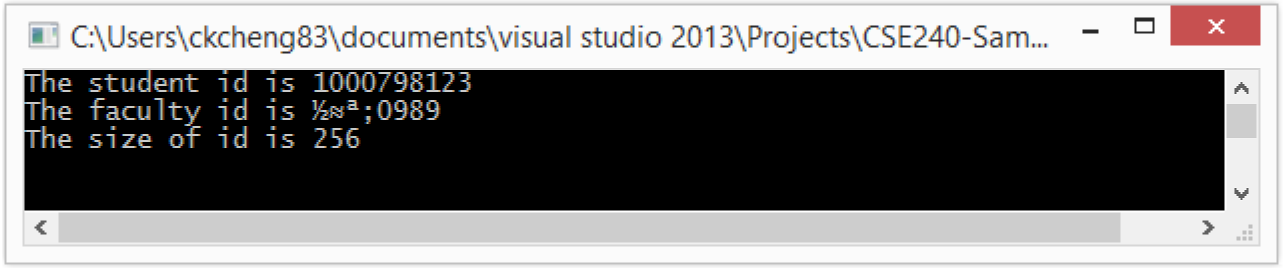
```
union type_name{
        type1   element1;
        type2   element2;
        …
        typen   elementn;
};
```

Why is union useful?

How is memory allocated for a union?

## UNION EXAMPLE

```c
#include <stdio.h>
#include <string.h>

union id {
    int student;
    char faculty[256];
};
union id personnel;

void main() {
     strcpy(personnel.faculty, "A1120989");
     personnel.student = 1000798123;
     printf("The student id is %d\n", personnel.student);
     printf("The faculty id is %s\n", personnel.faculty);
     printf("The size of id is %d", sizeof(personnel));
}
```


C:\Users\ckcheng83\documents\visual studio 2013\Projects\CSE240-Sam...

```
The student id is 1000798123
The faculty id is ½≈ª;0989
The size of id is 256
```

13

# PARAMETERS, SCOPE, AND POINTERS

## FUNCTIONS AND PARAMETER PASSING

A function/method/procedure/subroutine is a named block of code that must be explicitly called.

The purpose of functions are twofold: abstraction and reusability.

- **Abstraction:** statements that form a conceptual unit.
- **Reusability:** statements that can be executed in more than one place in the program.

A C program is ultimately a composition of such functions.

Functions communicate with the rest of the program by:

  - Either changing the global/static variables
  - Using parameters and/or return values

- In the declaration of a function, *formal parameters* are given, which are local variables of the function.
- When calling a function, *actual parameters* (also known as *arguments*) are given. Actual parameters are variables/values of the caller.

## PARAMETER PASSING

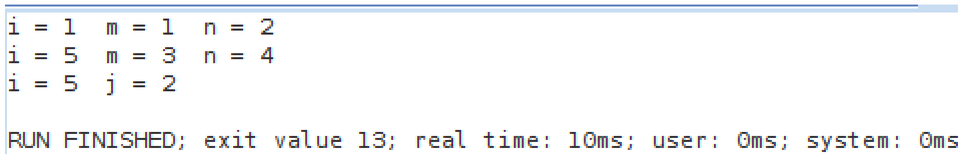
Different languages have different ways of implementing parameter passing. C has two general mechanisms:

- Pass-by-value: a formal parameter is a local variable in the function. It is initialized to the value of the actual parameter. It is a copy of the actual parameter. The copy will exist as an entry in a new function stack frame.
  - Advantage: no side-effects (safe and reliable).
  - Disadvantage: less flexible/less powerful.

- Pass-by-address (pointer): the formal parameter is a pointer to the actual parameter. The address will be stored as entry in the stack frame, but may point to both stack or heap memory. (This is technically known as *pass-by-reference* implemented by using pointers with pass-by-value.)

## PASS-BY-VALUE

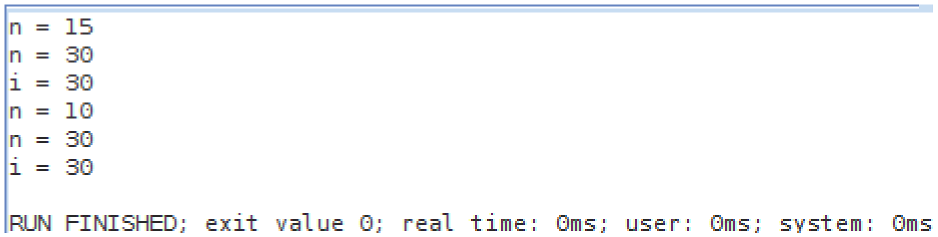
```c
#include <stdio.h>

int i = 1;                          // i is a global variable

void func(int m, int n) {           // m and n are formal parameters
    printf("i = %d  m = %d  n = %d\n", i, m, n);
    i = 5; m = 3; n = 4;            // Modify i, m and n.
    printf("i = %d  m = %d  n = %d \n", i, m, n);
}

void main() {
    int j = 2;                      // j is a local variable
    func(i, j);                     // i and j are actual parameters
    printf("i = %d  j = %d\n", i, j);
}
```



```
i = 1  m = 1  n = 2
i = 5  m = 3  n = 4
i = 5  j = 2

RUN FINISHED; exit value 13; real time: 10ms; user: 0ms; system: 0ms
```

## PASS-BY-VALUE (WITH A POINTER)

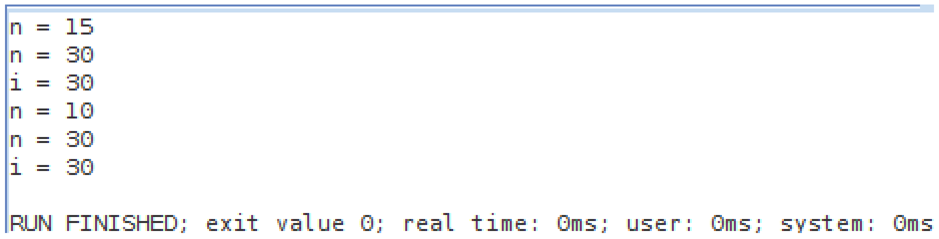
```c
#include <stdio.h>

void func(int *n) {
    printf("n = %d\n", *n);
    *n = 30;
    printf("n = %d\n", *n);
    n = 0;        // if before printf: error!
}                 // n is local, but *n is not

void main() {
    int* i = (int*)malloc(sizeof(int));
    *i = 15;
    func(i);
    printf("i = %d\n", *i);
    *i = 10;
    func(i);
    printf("i = %d\n", *i);
    free(i)
}
```



```
n = 15
n = 30
i = 30
n = 10
n = 30
i = 30

RUN FINISHED; exit value 0; real time: 0ms; user: 0ms; system: 0ms
```

# PASS-BY-REFERENCE

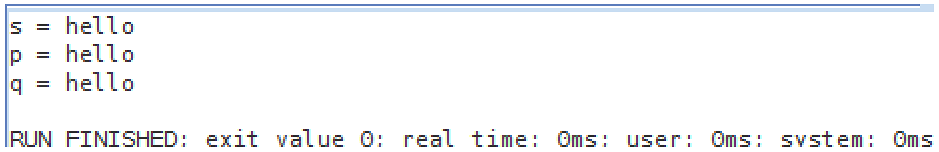
```c
#include <stdio.h>

void func(int *n) {
    printf("n = %d\n", *n);
    *n = 30;
    printf("n = %d\n", *n);
    n = 0;      // if before printf: error!
}               // n is local, but *n is not

void main() {
    int i = 15;
    func(&i);
    printf("i = %d\n", i);
    i = 10;
    func(&i);
    printf("i = %d\n", i);
}
```



```
n = 15
n = 30
i = 30
n = 10
n = 30
i = 30

RUN FINISHED; exit value 0; real time: 0ms; user: 0ms; system: 0ms
```

## PASSING STRING ADDRESS EXAMPLE

```c
#include <stdio.h>
#include <string.h>

char* getString(char *str) {
    return str;
}

void setString(char *str1, char *str2) {
    strcpy(str1, str2);
}

void main() {
    char *p, q[] = "morning", *s = "hello";
    printf("s = %s\n", s);
    p = getString(s);
    printf("p = %s\n", p);
    setString(q, p);
    printf("q = %s\n", q);
}
```



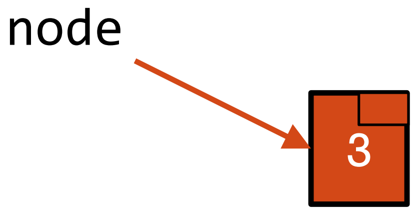
```
s = hello
p = hello
q = hello

RUN FINISHED: exit value 0: real time: 0ms: user: 0ms: system: 0ms
```

# LINKED LISTS

## IMPLEMENTING NODES

As a first step to creating a list, we need to implement a node struct.

- What is a node?
- What is a list?

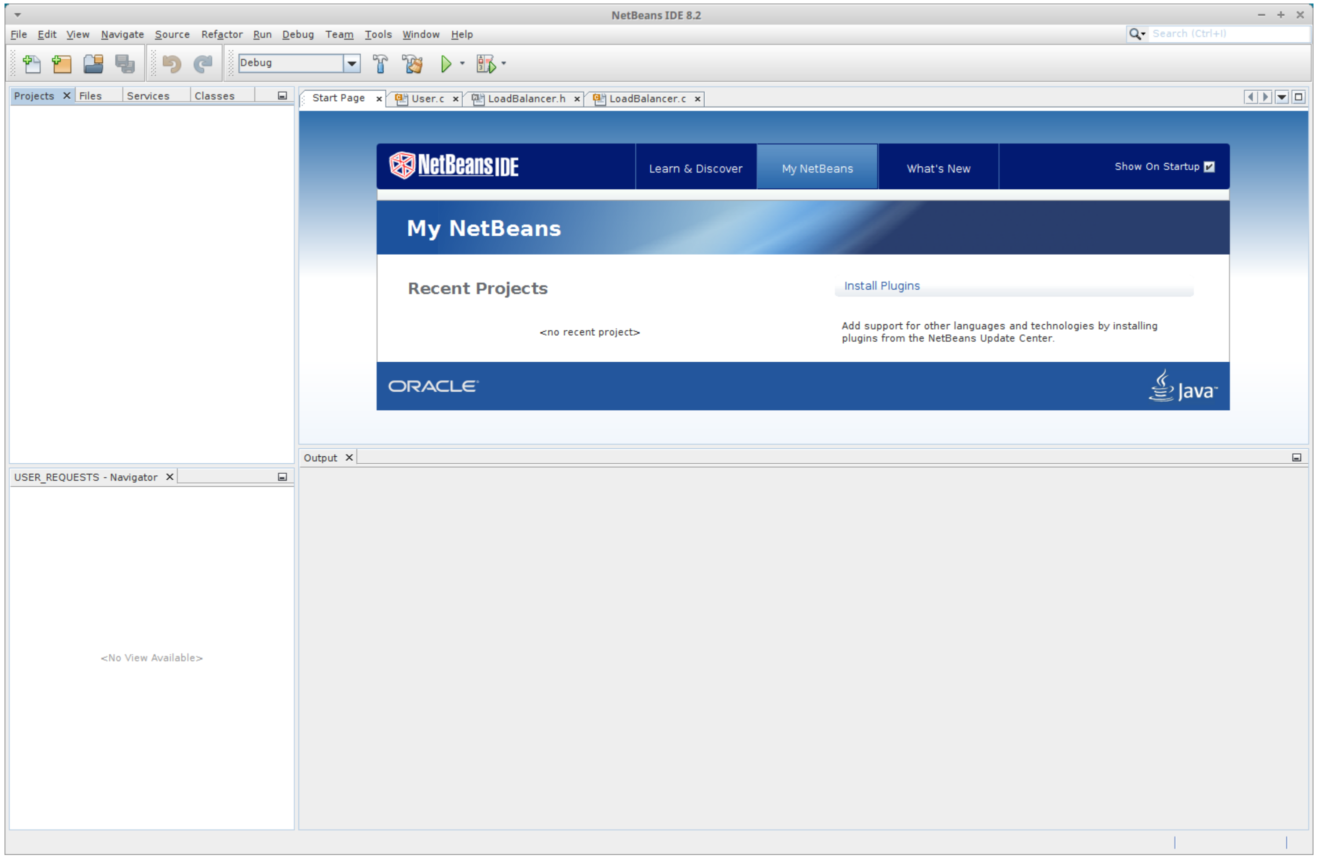


Some design considerations:

- How can we link nodes together?
- How can we represent an empty list?

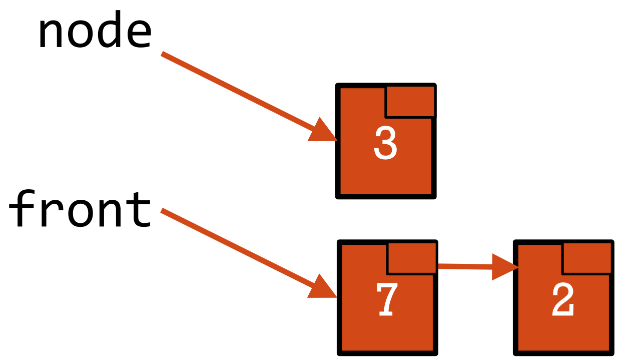
Next, let's try implementing a node, and then using it to store something.

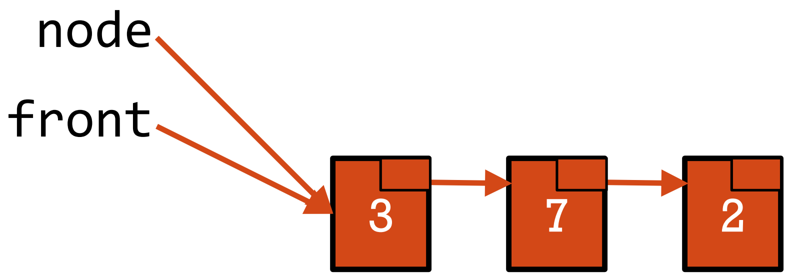
# IMPLEMENTING NODES


NetBeans IDE 8.2
File Edit View Navigate Source Refactor Run Debug Team Tools Window Help
Search (Ctrl+I)
Debug
Projects Files Services Classes
Start Page User.c LoadBalancer.h LoadBalancer.c
NetBeans IDE
Learn & Discover
My NetBeans
What's New
Show On Startup
My NetBeans
Recent Projects
<no recent project>
Install Plugins
Add support for other languages and technologies by installing plugins from the NetBeans Update Center.
ORACLE
Java
USER_REQUESTS - Navigator
<No View Available>
Output

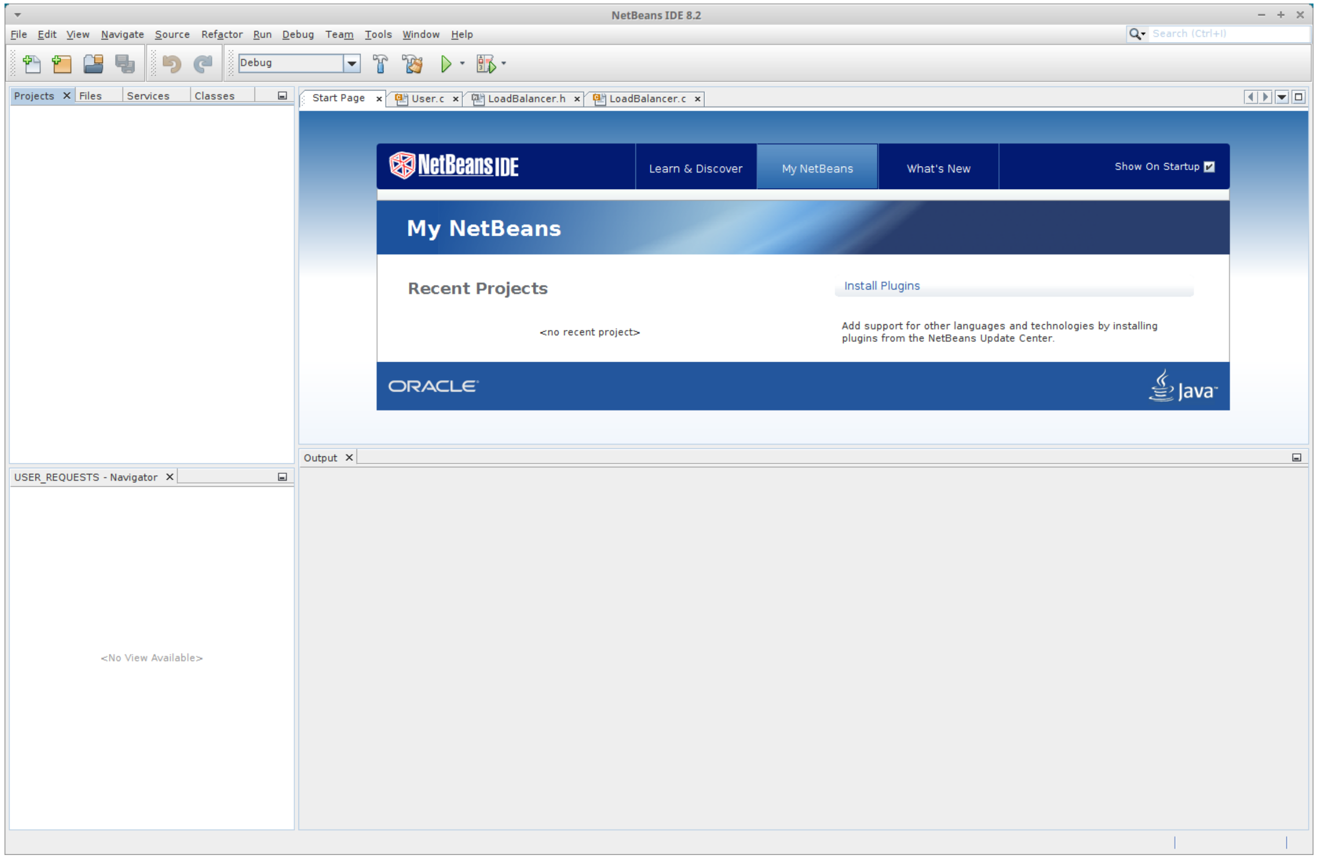
## ADDING TO THE FRONT

- Given that we have a node, and can populate it with data, we need to be able to add a node to an existing list of nodes.
- Let's try understanding the simplest case, where we want to add to the front of a list.
  - We'll need a pointer to a node, and to the front most node of an existing list.





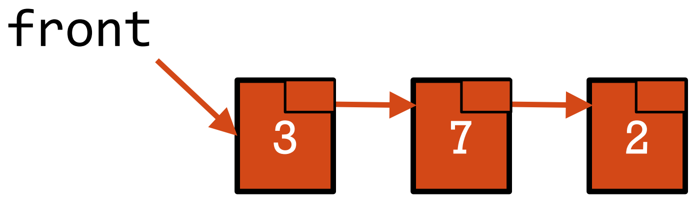
# ADDING TO THE FRONT

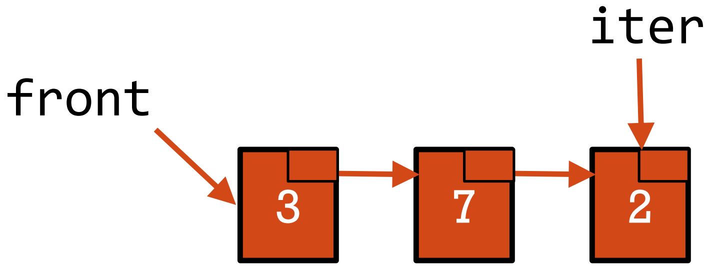


Add support for other languages and technologies by installing plugins from the NetBeans Update Center.

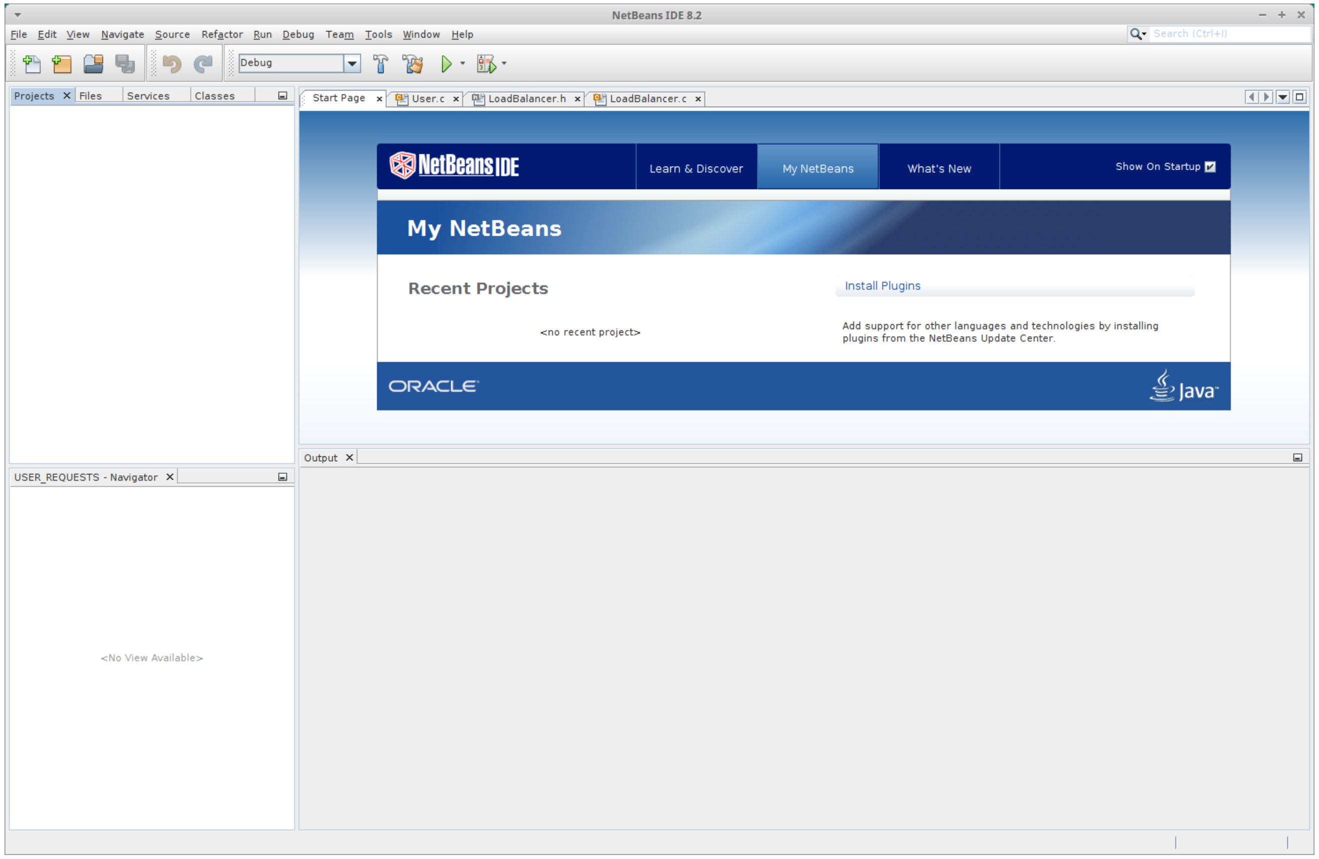
## TRAVERSAL

- Now we are able to construct a list and add as nodes as we want. However, this is only storing data. How do we view data?
- Again, let's try the simplest case, where we want to visit each node and perform an operating like displaying it.
    - We'll need a pointer to the front most node of an existing list.



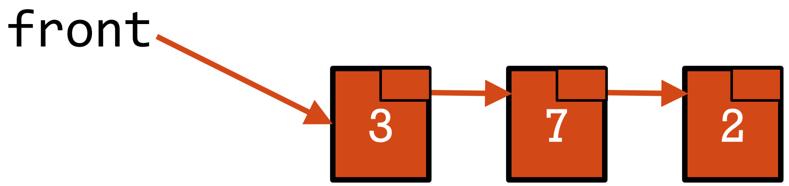


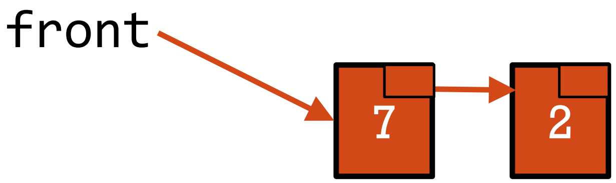
## TRAVERSAL


NetBeans IDE 8.2
File Edit View Navigate Source Refactor Run Debug Team Tools Window Help
Search (Ctrl+I)
Debug
Projects Files Services Classes
Start Page User.c LoadBalancer.h LoadBalancer.c
NetBeans IDE
Learn & Discover
My NetBeans
What's New
Show On Startup
My NetBeans
Recent Projects
<no recent project>
Install Plugins
Add support for other languages and technologies by installing plugins from the NetBeans Update Center.
ORACLE
Java
Output
USER_REQUESTS - Navigator
<No View Available>

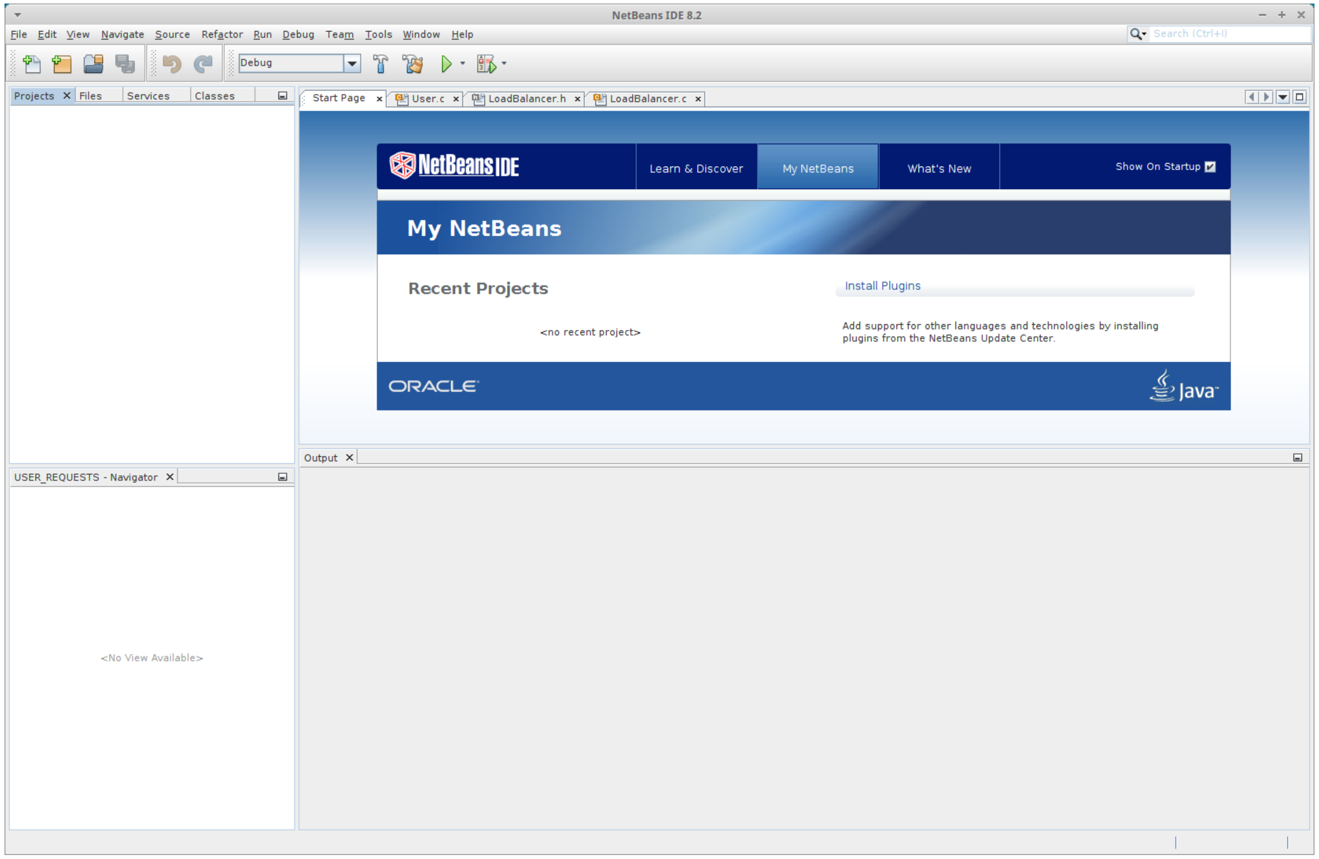
## REMOVING FROM THE FRONT

- Given that we can traverse a list, and view its contents, we now want to look at removing nodes from it.
- Let's try understanding the simplest case, where we want to remove from the front of a list.
  - We'll need a pointer to the front most node of an existing list.



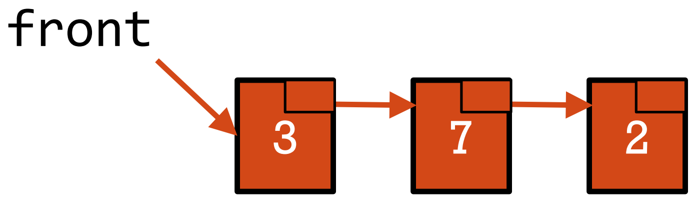


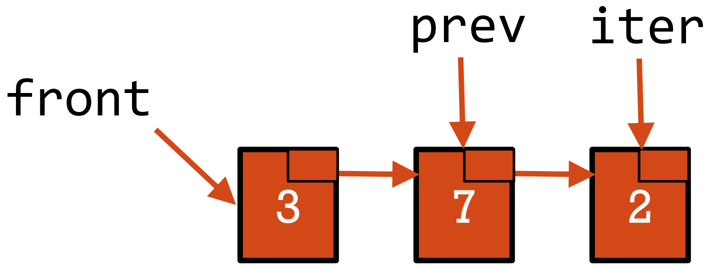
# REMOVING FROM THE FRONT


NetBeans IDE 8.2
File Edit View Navigate Source Refactor Run Debug Team Tools Window Help
Search (Ctrl+I)
Debug
Projects Files Services Classes
Start Page User.c LoadBalancer.h LoadBalancer.c
NetBeans IDE
Learn & Discover
My NetBeans
What's New
Show On Startup
My NetBeans
Recent Projects
<no recent project>
Install Plugins
Add support for other languages and technologies by installing plugins from the NetBeans Update Center.
ORACLE
Java
Output
USER_REQUESTS - Navigator
<No View Available>

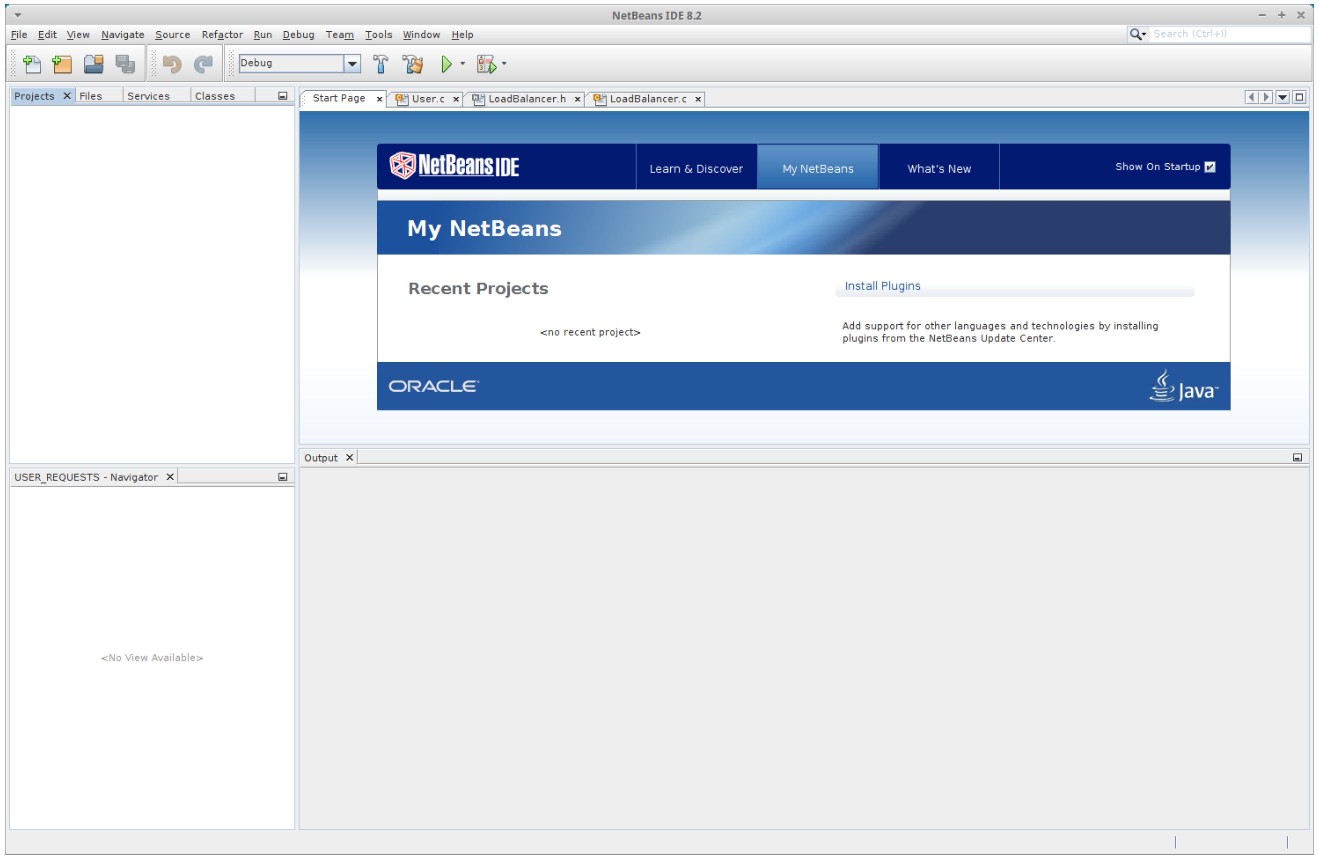
## RETRIEVING A NODE

- As an extension of traversing a list, let's write a function to select a particular node and return it.
  - We'll need a pointer to the front most node of an existing list.
- Now suppose for a moment that we wanted to be able to select nodes for the purpose of editing them or removing them.
- It's not enough to return the node, we need return the previous node.





## RETRIEVING A NODE


NetBeans IDE 8.2
File Edit View Navigate Source Refactor Run Debug Team Tools Window Help
Search (Ctrl+I)
Debug
Projects Files Services Classes
Start Page User.c LoadBalancer.h LoadBalancer.c
NetBeans IDE
Learn & Discover
My NetBeans
What's New
Show On Startup
My NetBeans
Recent Projects
<no recent project>
Install Plugins
Add support for other languages and technologies by installing plugins from the NetBeans Update Center.
ORACLE
Java
Output
USER_REQUESTS - Navigator
<No View Available>