Com S // Cpr E // Math 5250

Numerical Analysis of High-Performance Computing

Instructor: Songting Luo

Lecture 11: More on Pointers, Makefiles, and Linked Lists

Outline

1. More on Pointers
2. Makefiles
3. Linked Lists

More on Pointers

Code to swap integers, code to swap pointers

$ISUHPC/lectures/lecture11/codes/more_pointers/pointer_wrong.c:

```
1  #include <stdio.h>
2
3  void swap_nums(int *x, int *y)
4  {
5      int tmp;
6      tmp = *x;
7      *x = *y;
8      *y = tmp;
9  }
10
11 void swap_pointers(char *x, char *y)
12 {
13     char *tmp;
14     tmp = x;
15     x = y;
16     y = tmp;
17 }
```

Code to swap integers, code to swap pointers

```
1  int main()
2  {
3      int a = 3;
4      int b = 4;
5      swap_nums(&a,&b);
6      printf("a is %d\n", a);
7      printf("b is %d\n", b);
8
9      char* s1 = "I should print second";
10     char* s2 = "I should print first";
11     swap_pointers(s1,s2);
12     printf("s1 is %s\n", s1);
13     printf("s2 is %s\n", s2);
14
15     return 0;
16 }
```

Code to swap integers, code to swap pointers

```c
int main()
{
    int a = 3;
    int b = 4;
    swap_nums(&a,&b);
    printf("a is %d\n", a);
    printf("b is %d\n", b);

    char* s1 = "I should print second";
    char* s2 = "I should print first";
    swap_pointers(s1,s2);
    printf("s1 is %s\n", s1);
    printf("s2 is %s\n", s2);

    return 0;
}
```

```
$ gcc pointer_wrong.c
$ a.out
a is 4
b is 3
s1 is I should print second
s2 is I should print first
```

What’s wrong this code?

Pointer to a pointer

```c
#include <stdio.h>

void swap_nums(int *x, int *y)
{
    int tmp;
    tmp = *x;
    *x = *y;
    *y = tmp;
}

void swap_pointers(char **x, char **y)
{
    char *tmp;  // temporary pointer
    tmp = *x;   // a dereferenced pointer-pointer is a pointer
    *x = *y;    // set x pointer to y pointer
    *y = tmp;   // set y pointer to x pointer
}
```

NOTE: if you want to change the pointer – and not the value that it points to – you need to pass the pointer by reference: you need to pass the pointer to the pointer.

Pointer to a pointer

```
1  int main()
2  {
3      int a = 3;
4      int b = 4;
5      swap_nums(&a,&b);
6      printf("a is %d\n", a);
7      printf("b is %d\n", b);
8
9      char* s1 = "I should print second";
10     char* s2 = "I should print first";
11     swap_pointers(&s1,&s2);
12     printf("s1 is %s\n", s1);
13     printf("s2 is %s\n", s2);
14
15     return 0;
16 }
```

Pointer to a pointer

```c
int main()
{
    int a = 3;
    int b = 4;
    swap_nums(&a,&b);
    printf("a is %d\n", a);
    printf("b is %d\n", b);

    char* s1 = "I should print second";
    char* s2 = "I should print first";
    swap_pointers(&s1,&s2);
    printf("s1 is %s\n", s1);
    printf("s2 is %s\n", s2);

    return 0;
}
```

```
$ gcc pointer_wrong.c
$ a.out
a is 4
b is 3
s1 is I should print first
s2 is I should print second
```

Makefiles

Makefiles

Makefile

A Makefile is a common way of automating software builds and other complex tasks with dependencies. A Makefile is itself a program in a special language.

```
1  # $ISUHPC/lectures/lecture11/codes/fraction/Makefile
2  output.txt: main.exe
3          ./main.exe > output.txt
4
5  main.exe: main.o fraction_add.o fraction_reduce.o \
6          get_prime_factors.o
7          gcc fraction_add.o fraction_reduce.o \
8          get_prime_factors.o main.o -o main.exe
9
10 main.o: main.c
11         gcc -c main.c
12 fraction_add.o: fraction_add.c
13         gcc -c fraction_add.c
14 fraction_reduce.o: fraction_reduce.c
15         gcc -c fraction_reduce.c
16 get_prime_factors.o: get_prime_factors.c
17         gcc -c get_prime_factors.c
```

Compiling multiple functions

```
$ cd $ISUHPC/lectures/lecture11/codes/fraction/
$ rm -f *.o *.exe *.txt  # remove old versions

$ make main.exe
gcc -c main.c
gcc -c fraction_add.c
gcc -c fraction_reduce.c
gcc -c get_prime_factors.c
gcc fraction_add.o fraction_reduce.o \
        get_prime_factors.o main.o -o main.exe
```

Uses commands for making `main.exe`.

NOTE: first had to make all the `.o` files. Then executed the rule to make `main.exe`.

Structure of a Makefile

Typical element in the simple Makefile:

```
target:   dependencies
<TAB>   command(s) to make target
```

Important to use tab character, not spaces !!!

Warning: some editors replace tabs with spaces!

Typing “make target” means:

1. Make sure all the dependencies are up to date (those that are also targets)
2. If target is older than any dependency, recreate it using specified commands.

NOTE: these rules are applied recursively.

Compiling multiple functions

```
$ rm -f *.o *.exe *.txt  # remove old versions

$ make fraction_add.o
gcc -c fraction_add.c

$ make get_prime_factors.o
gcc -c get_prime_factors.c

$ make main.o
gcc -c main.c

$ make main.exe
gcc -c fraction_reduce.c
gcc fraction_add.o fraction_reduce.o \
        get_prime_factors.o main.o -o main.exe
```

NOTE: Typing `make main.exe` required us to compile `fraction_reduce.c`, but not the other object files.

Compiling multiple functions

```
$ ls -l fraction_add.*
-rw-r--r--  1 luos staff  375 Feb 23 20:30 fraction_add.c
-rw-r--r--  1 luos staff  920 Feb 23 22:31 fraction_add.o

$ make fraction_add.o
make: ‘fraction_add.o’ is up to date.

$ touch fraction_add.c; ls -l fraction_add.c
-rw-r--r--  1 luos staff  375 Feb 23 22:34 fraction_add.c

$ make main.exe
gcc -c fraction_add.c
gcc fraction_add.o fraction_reduce.o \
        get_prime_factors.o main.o -o main.exe
```

Implicit rules

First version of Makefile has 4 rules that are very similar:

```
1  # $ISUHPC/lectures/lecture11/codes/fraction/Makefile
2  output.txt: main.exe
3          ./main.exe > output.txt
4
5  main.exe: main.o fraction_add.o fraction_reduce.o \
6          get_prime_factors.o
7          gcc fraction_add.o fraction_reduce.o \
8          get_prime_factors.o main.o -o main.exe
9
10 main.o: main.c
11         gcc -c main.c
12 fraction_add.o: fraction_add.c
13         gcc -c fraction_add.c
14 fraction_reduce.o: fraction_reduce.c
15         gcc -c fraction_reduce.c
16 get_prime_factors.o: get_prime_factors.c
17         gcc -c get_prime_factors.c
```

Replace these with a pattern rule ...

Implicit rules

General rule to make `.o` file from `.c` file:

```
1  # $ISUHPC/lectures/lecture11/codes/fraction/Makefile2
2  output.txt: main.exe
3          main.exe > output.txt
4
5  main.exe: main.o fraction_add.o fraction_reduce.o \
6          get_prime_factors.o
7          gcc fraction_add.o fraction_reduce.o \
8          get_prime_factors.o main.o -o main.exe
9
10 %.o: %.c
11         gcc -c $<
```

Rather than rule to make each separately, implicit rule (lines 10-11) is used for all 4

Implicit rules

General rule to make `.o` file from `.c` file:

```
1  # $ISUHPC/lectures/lecture11/codes/fraction/Makefile2
2  output.txt: main.exe
3          main.exe > output.txt
4
5  main.exe: main.o fraction_add.o fraction_reduce.o \
6          get_prime_factors.o
7          gcc fraction_add.o fraction_reduce.o \
8          get_prime_factors.o main.o -o main.exe
9
10 %.o: %.c
11         gcc -c $<
```

Rather than rule to make each separately, implicit rule (lines 10-11) is used for all 4

```
$ make main.exe -f Makefile2  # use -f to use different Makefile
gcc -c main.c
gcc -c fraction_add.c
gcc -c fraction_reduce.c
gcc -c get_prime_factors.c
gcc fraction_add.o fraction_reduce.o \
        get_prime_factors.o main.o -o main.exe
```

Defining a macro

We had to repeat the list of `.o` files twice:

```
# $ISUHPC/lectures/lecture11/codes/fraction/Makefile2
output.txt: main.exe
        main.exe > output.txt

main.exe: main.o fraction_add.o fraction_reduce.o \
        get_prime_factors.o
        gcc fraction_add.o fraction_reduce.o \
        get_prime_factors.o main.o -o main.exe

%.o: %.c
        gcc -c $<
```

Can simplify and reduce errors by defining a macro.

Defining a macro

```
1  # $ISUHPC/lectures/lecture11/codes/fraction/Makefile3
2
3  OBJECTS = main.o fraction_add.o fraction_reduce.o \
4            get_prime_factors.o
5
6  output.txt: main.exe
7            main.exe > output.txt
8
9  main.exe: $(OBJECTS)
10           gcc $(OBJECTS) -o main.exe
11
12 %.o: %.c
13           gcc -c $<
```

By convention, all-caps names are used for Makefile macros.

Note that to use `OBJECTS` we must write `$(OBJECTS)`.

Defining a macro

```
1  # $ISUHPC/lectures/lecture11/codes/fraction/Makefile4
2
3  CC = gcc
4  FFLAGS = -O3
5  LFFLAG =
6  OBJECTS = main.o fraction_add.o fraction_reduce.o \
7            get_prime_factors.o
8
9  output.txt: main.exe
10           main.exe > output.txt
11
12 main.exe: $(OBJECTS)
13           $(CC) $(LFLAGS) $(OBJECTS) -o main.exe
14
15 %.o: %.c
16           $(CC) $(FFLAGS) -c $<
```

Here we have added macros for the name of the C compiler (`$(CC)`) command and for the compile flags (`$(FFLAGS)`) and linking flags (`$(LFLAGS)`).

NOTE: `-O3` means that the compiler will try to optimize code with optimizer level 3.

Phony targets

```
1  # $ISUHPC/lectures/lecture11/codes/fraction/Makefile5
2  CC = gcc
3  FFLAGS = -O3
4  LFFLAG =
5  OBJECTS = main.o fraction_add.o fraction_reduce.o \
6            get_prime_factors.o
7
8  .PHONY: clean
9
10 output.txt: main.exe
11           main.exe > output.txt
12
13 main.exe: $(OBJECTS)
14           $(CC) $(LFLAGS) $(OBJECTS) -o main.exe
15
16 %.o: %.c
17           $(CC) $(FFLAGS) -c $<
18
19 clean:
20           rm -f $(OBJECTS) main.exe output.txt
```

Phony targets

```
1  # $ISUHPC/lectures/lecture11/codes/fraction/Makefile5
2  CC = gcc
3  FFLAGS = -O3
4  LFFLAG =
5  OBJECTS = main.o fraction_add.o fraction_reduce.o \
6            get_prime_factors.o
7
8  .PHONY: clean
9
10 output.txt: main.exe
11           main.exe > output.txt
12
13 main.exe: $(OBJECTS)
14           $(CC) $(LFLAGS) $(OBJECTS) -o main.exe
15
16 %.o: %.c
17           $(CC) $(FFLAGS) -c $<
18
19 clean:
20           rm -f $(OBJECTS) main.exe output.txt
```

```
$ make clean -f Makefile5
rm -f main.o fraction_add.o fraction_reduce.o get_prime_factors.o mai
```

make help

```
1  # $ISUHPC/lectures/lecture11/codes/fraction/Makefile6
2  CC = gcc
3  FFLAGS = -O3
4  LFFLAG =
5  OBJECTS = main.o fraction_add.o fraction_reduce.o \
6            get_prime_factors.o
7
8  .PHONY: clean help
9
10 output.txt: main.exe
11           main.exe > output.txt
12
13 main.exe: $(OBJECTS)
14           $(CC) $(LFLAGS) $(OBJECTS) -o main.exe
15
16 %.o: %.c
17           $(CC) $(FFLAGS) -c $<
18
19 clean:
20           rm -f $(OBJECTS) main.exe output.txt
```

make help

```
1  help:
2          @echo "Valid targets:"
3          @echo "  main.exe"
4          @echo "  main.o"
5          @echo "  fraction_add.o"
6          @echo "  fraction_reduce.o"
7          @echo "  get_primes.o"
8          @echo "  clean:  removes *.o, *.txt, and *.exe files"
```

```
$ make clean -f Makefile6
rm -f main.o fraction_add.o fraction_reduce.o get_prime_factors.o mai

$ make help -f Makefile6
Valid targets:
main.exe
main.o
fraction_add.o
fraction_reduce.o
get_primes.o
clean:  removes *.o, *.txt, and *.exe files
```

Linked Lists

Linked lists

$ISUHPC/lectures/lecture11/codes/linked_list/node.h:

```
1  #ifndef __NODE_H__
2  #define __NODE_H__
3
4  typedef struct node node;
5  struct node
6  {
7     int position;
8     int value;
9     node* next;
10 };
11
12 // Functions associated with struct node
13 int GetNumberOfNodes();
14 void GenerateList(node** head,const int num);
15 void Print(const int forward,const node* head);
16 void PrintList(const node* head);
17 void ReversePrintList(const node* head);
18 int GetKey();
19 void SearchList(const node* head,const int key);
20 void DeleteList(node** head);
21
22 #endif
```

Linked Lists

$ISUHPC/lectures/lecture11/codes/linked_list/main.c:

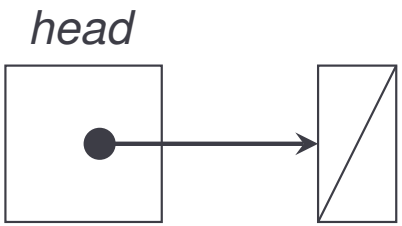
```
1  int main()
2  {
3     // Declare the head node in my list
4     node* head = NULL;
5
6     // Set number of nodes and generate a new list
7     const int num_nodes = GetNumberOfNodes();
8     GenerateList(&head,num_nodes);
9
10    // Print list to screen
11    Print(1,head); // foward print
12    Print(0,head); // reverse print
13
14    // Ask for a key, then search list
15    if(num_nodes>0)
16    {
17       const int key = GetKey();
18       SearchList(head,key);
19    }
20
21    // Delete list (free up memory)
22    DeleteList(&head);
23 }
```

Linked Lists

Example: Add 3 nodes to the initially empty list. First node has a value of **3**, second node has a value of **11**, and third node has a value of **7**.

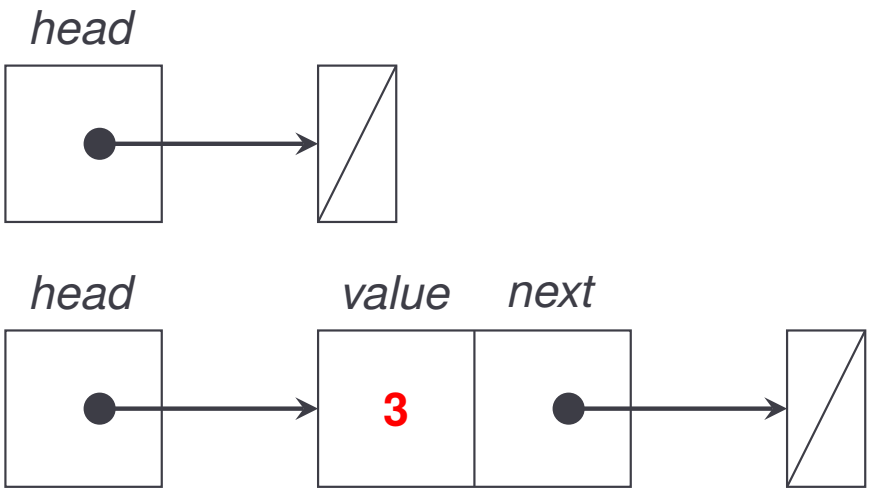
Linked Lists

Example: Add 3 nodes to the initially empty list. First node has a value of **3**, second node has a value of **11**, and third node has a value of **7**.



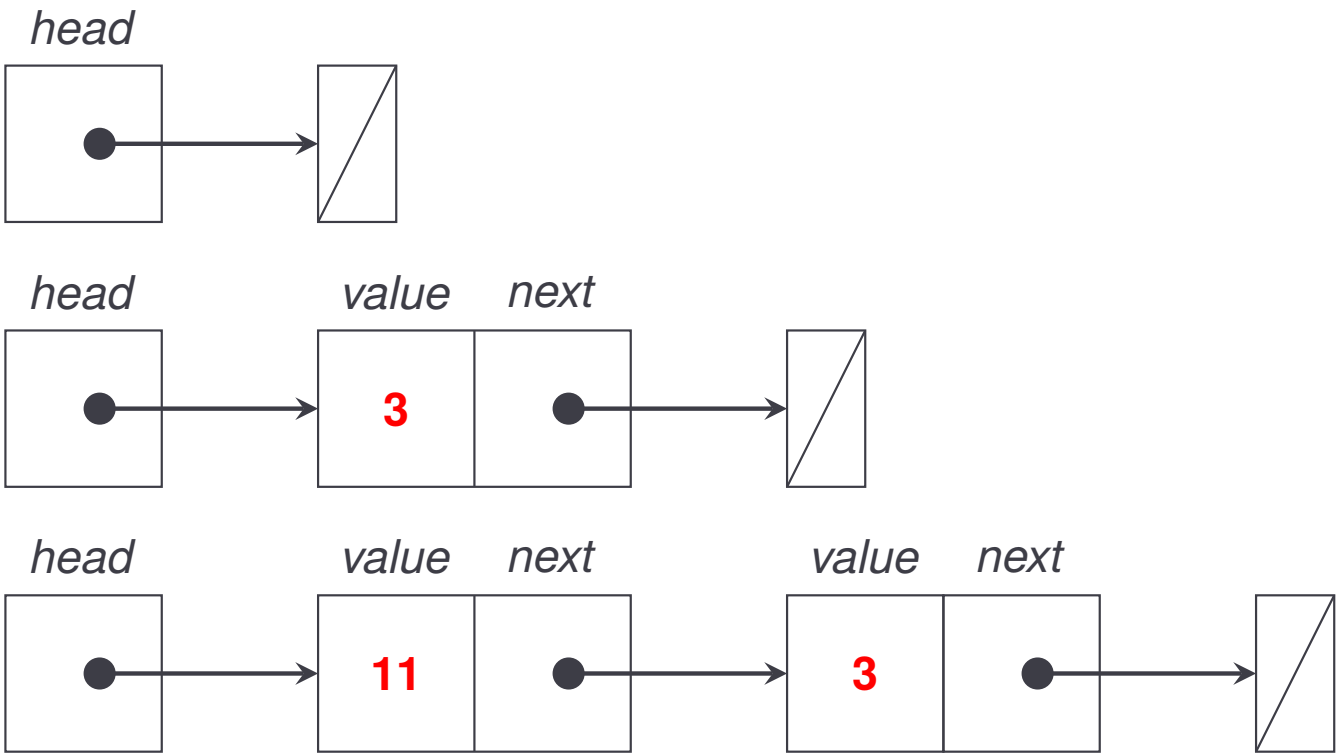
Linked Lists

Example: Add 3 nodes to the initially empty list. First node has a value of **3**, second node has a value of **11**, and third node has a value of **7**.



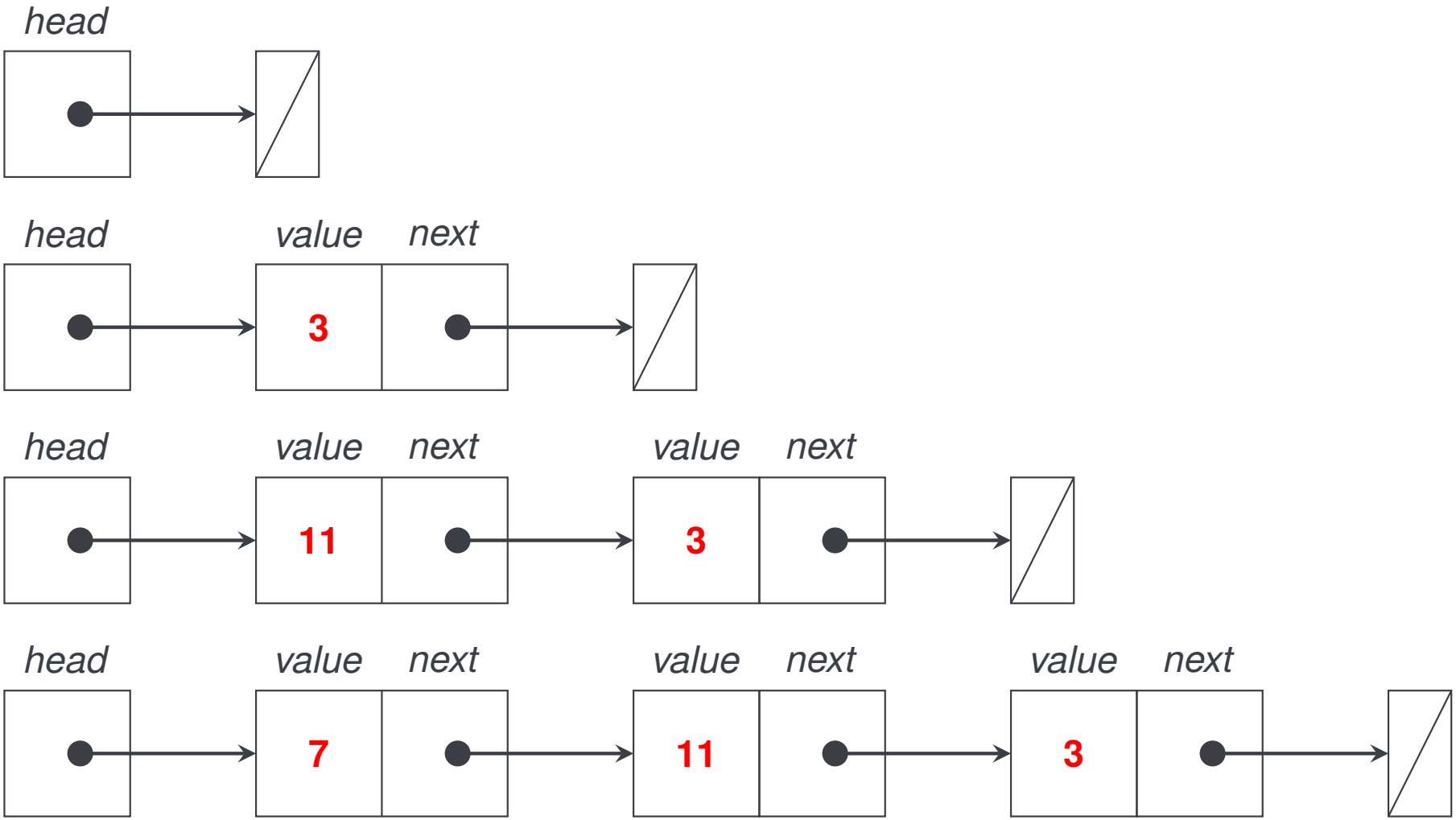
Linked Lists

Example: Add 3 nodes to the initially empty list. First node has a value of **3**, second node has a value of **11**, and third node has a value of **7**.



Linked Lists

Example: Add 3 nodes to the initially empty list. First node has a value of **3**, second node has a value of **11**, and third node has a value of **7**.



Linked lists

$ISUHPC/lectures/lecture11/codes/linked_list/GenerateList.c:

```
1  void GenerateList(node** head, const int num_nodes)
2  {
3     node* temp; srand( time(NULL) );
4
5     for (int i=0; i<num_nodes; i++)
6     {
7        temp = (node*)malloc(sizeof(node));
8        temp->value = rand()%num_nodes; temp->position = 0;
9        printf("%4i",temp->value);
10
11       if (*head == NULL)
12       {
13          *head = temp;
14          (*head)->next = NULL;
15       }
16       else
17       {
18          temp->next = *head;
19          *head = temp;
20       }
21    }
```

Linked lists

$ISUHPC/lectures/lecture11/codes/linked_list/GenerateList.c:

```
    printf("\n");

    node* ptr = *head; int pos = 1;
    while(ptr!=NULL)
    {
        ptr->position = pos;
        pos = pos+1;
        ptr = ptr->next;
    }
}
```

Linked lists

$ISUHPC/lectures/lecture11/codes/linked_list/Print.c:

```
1  void Print(const int forward, const node* head)
2  {
3     if (head==NULL)
4     { printf(" List is emtpy.\n\n"); return; }
5     printf("\n");
6     printf(" ----------------------------------------\n");
7     printf(" |Pos:|Val:|    Address:    |      Next:     |\n");
8     printf(" ----------------------------------------\n");
9     switch(forward)
10    {
11       case 0:
12          ReversePrintList(head);
13          break;
14       case 1:
15          PrintList(head);
16          break;
17       default:
18          printf("\n Error: forward must be 0 or 1.\n");
19          printf("  forward = %i\n",forward); exit(1);
20    }
21    printf(" ----------------------------------------\n");
22 }
```

Linked lists

$ISUHPC/lectures/lecture11/codes/linked_list/Print.c:

```
1  void PrintLine(const int pos,const int val,
2                 const node* head,const node* next)
3  { printf(" |%3i |%3i |%15p |%15p |\n",pos,val,head,next); }
4
5  void PrintList(const node* head)
6  {
7     PrintLine(head->position,head->value,head,head->next);
8     if (head->next == NULL)
9     { return; }
10    PrintList(head->next);
11 }
12
13 void ReversePrintList(const node* head)
14 {
15    if (head->next == NULL)
16    {
17       PrintLine(head->position,head->value,head,head->next);
18       return;
19    }
20    ReversePrintList(head->next);
21    PrintLine(head->position,head->value,head,head->next);
22 }
```

Linked lists

$ISUHPC/lectures/lecture11/codes/linked_list/GetKey.c:

```c
#include <stdio.h>

int GetKey()
{
    int key;
    printf("\n Enter key to search: ");
    scanf("%i", &key);

    return key;
}
```

Linked lists

$ISUHPC/lectures/lecture11/codes/linked_list/SearchList.c:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "node.h"
4
5  void SearchList(const node* head,
6                  const int key)
7
8  {
9     if (head->value==key)
10    {
11       printf(" Key found at Position: %i\n",
12              head->position);
13    }
14
15    if (head->next==NULL)
16    {
17       printf("\n"); return;
18    }
19
20    SearchList(head->next,key);
21 }
```

Linked lists

$ISUHPC/lectures/lecture11/codes/linked_list/DeleteList.c:

```
1  #include <stdlib.h>
2  #include "node.h"
3
4  void DeleteList(node** head)
5  {
6     node* temp;
7     while (*head!=NULL)
8     {
9        temp = *head;
10       *head = (*head)->next;
11       free(temp);
12    }
13 }
```

Linked lists

$ISUHPC/lectures/lecture11/codes/linked_list/Makefile:

```
1  # $ISUHPC/lectures/lecture11/codes/linked_list/Makefile
2  CC = gcc
3  FFLAGS = -O3 -Wall
4  LFFLAG =
5  OBJECTS = main.o \
6  GetNumberOfNodes.o \
7  GenerateList.o \
8  Print.o \
9  GetKey.o \
10 SearchList.o \
11 DeleteList.o
12
13 .PHONY: clean help
14
15 main.exe: $(OBJECTS)
16         $(CC) $(LFLAGS) $(OBJECTS) -o main.exe
17
18 %.o: %.c
19         $(CC) $(FFLAGS) -c $<
20
21 clean:
22         rm -f $(OBJECTS) main.exe
```

Linked lists

$ISUHPC/lectures/lecture11/codes/linked_list/Makefile:

```
help:
        @echo "Valid targets:"
        @echo "  main.exe"
        @echo "  main.o"
        @echo "  GetNumberOfNodes.o"
        @echo "  GenerateList.o"
        @echo "  Print.o"
        @echo "  GetKey.o"
        @echo "  SearchList.o"
        @echo "  DeleteList.o"
        @echo "  clean:  removes *.o and *.exe files"
```

```
$ make
gcc -O3 -Wall -c main.c
gcc -O3 -Wall -c GetNumberOfNodes.c
gcc -O3 -Wall -c GenerateList.c
gcc -O3 -Wall -c Print.c
gcc -O3 -Wall -c GetKey.c
gcc -O3 -Wall -c SearchList.c
gcc -O3 -Wall -c DeleteList.c
gcc  main.o GetNumberOfNodes.o GenerateList.o Print.o GetKey.o
SearchList.o DeleteList.o -o main.exe
```

Linked lists

```
 $ main.exe

  Enter the number of nodes: 10

  0   9   4   9   1   3   5   9   0   2

-------------------------------------------
|Pos:|Val:|    Address:    |      Next:     |
-------------------------------------------
|  1 |  2 | 0x7ffe20404d30 | 0x7ffe20404d20 |
|  2 |  0 | 0x7ffe20404d20 | 0x7ffe20404d10 |
|  3 |  9 | 0x7ffe20404d10 | 0x7ffe20404d00 |
|  4 |  5 | 0x7ffe20404d00 | 0x7ffe20404cf0 |
|  5 |  3 | 0x7ffe20404cf0 | 0x7ffe20404ce0 |
|  6 |  1 | 0x7ffe20404ce0 | 0x7ffe20404cd0 |
|  7 |  9 | 0x7ffe20404cd0 | 0x7ffe20404cc0 |
|  8 |  4 | 0x7ffe20404cc0 | 0x7ffe20404cb0 |
|  9 |  9 | 0x7ffe20404cb0 | 0x7ffe20404ca0 |
| 10 |  0 | 0x7ffe20404ca0 |            0x0 |
-------------------------------------------
```

Linked lists

```
-------------------------------------------
|Pos:|Val:|    Address:    |      Next:     |
-------------------------------------------
| 10 |  0 | 0x7ffe20404ca0 |            0x0 |
|  9 |  9 | 0x7ffe20404cb0 | 0x7ffe20404ca0 |
|  8 |  4 | 0x7ffe20404cc0 | 0x7ffe20404cb0 |
|  7 |  9 | 0x7ffe20404cd0 | 0x7ffe20404cc0 |
|  6 |  1 | 0x7ffe20404ce0 | 0x7ffe20404cd0 |
|  5 |  3 | 0x7ffe20404cf0 | 0x7ffe20404ce0 |
|  4 |  5 | 0x7ffe20404d00 | 0x7ffe20404cf0 |
|  3 |  9 | 0x7ffe20404d10 | 0x7ffe20404d00 |
|  2 |  0 | 0x7ffe20404d20 | 0x7ffe20404d10 |
|  1 |  2 | 0x7ffe20404d30 | 0x7ffe20404d20 |
-------------------------------------------

Enter key to search: 9
Key found at Position: 3
Key found at Position: 7
Key found at Position: 9
```

Lab assignment: linked lists

- Develop a program (with multiple files) for linked list with application for polynomial representation, addition and evaluation. Use Makefile.
- Update Git repository
- Submit source codes and screenshots.