

CH-230-A

Programming in C and C++

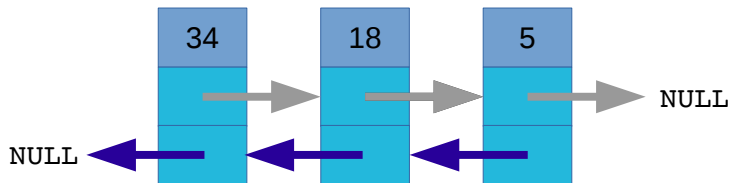
C/C++

Lecture 7

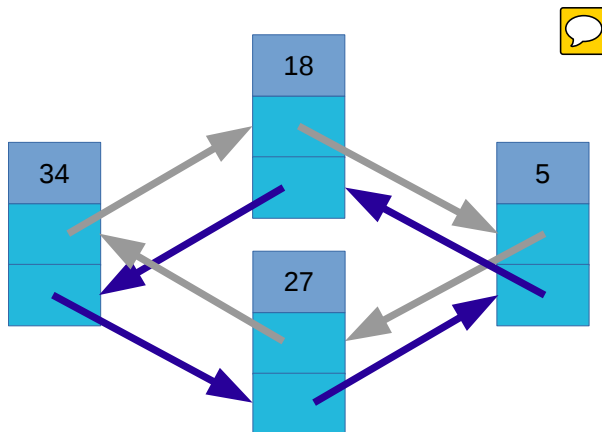
Dr. Kinga Lipskoch

Fall 2020

Doubly Linked Lists



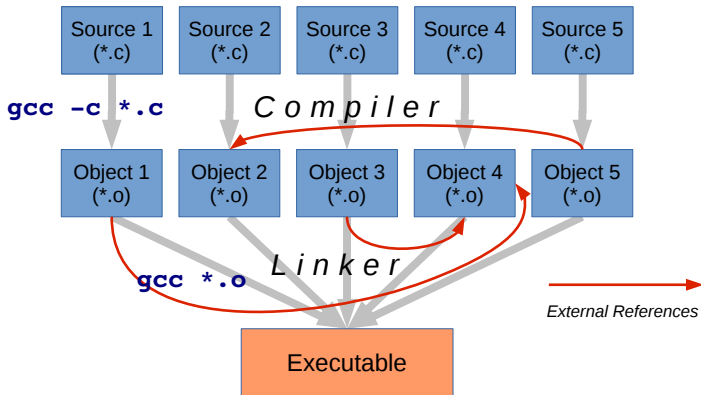
Circular Doubly Linked Lists



Building from Multiple Sources

- ▶ C compilers can compile multiple sources files into one executable
- ▶ For every declaration there must be one definition in one of the compiled files
 - ▶ Indeed also libraries play a role
 - ▶ This control is performed by the linker
- ▶ `gcc -o name file1.c file2.c file3.c`

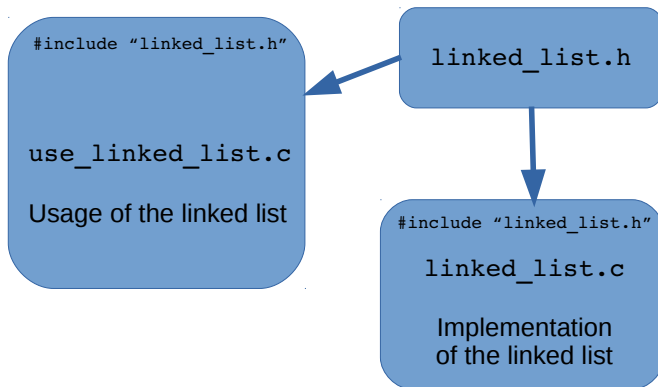
Linking



Linked List Header File

```
1  /*****
2  *
3  * A simply linked list is linked from node structures
4  * whose size can grow as needed. Adding more elements
5  * to the list will just cause it to grow and removing
6  * elements will cause it to shrink.
7  *-----*
8  * struct ll_node
9  *     used to hold the information for a node of a
10 *     simply linked list
11 *-----*
12 * Function declaration (routines)
13 *
14 *     push_front -- add an element in the beginning
15 *     push_back  -- add an element in the end
16 *     dispose_list -- remove all the elements
17 *     ...
18 *****/
```

Definition Import via #include



Compile Linked List from Multiple Sources

- ▶ Create a project with your IDE, add all files including the header file and then compile and execute
- ▶ or
- ▶ **Compile:** `gcc -Wall -o use_linked_list linked_list.c use_linked_list.c`
- ▶ **Execute:** `./use_linked_list`

Cygwin

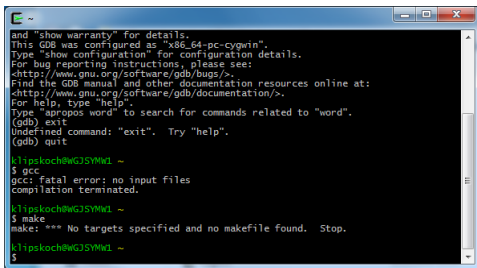
- ▶ Cygwin is a Unix-like environment and command-line interface for Microsoft Windows
- ▶ Cygwin provides native integration of Windows-based applications, data, and other system resources with applications, software tools, and data of the Unix-like environment
- ▶ Thus it is possible to launch Windows applications from the Cygwin environment, as well as to use Cygwin tools and applications within the Windows operating context

- ▶ Go to <https://cygwin.com/install.html>, download setup-x86_64.exe and install it
- ▶ During installation add gdb, gcc-core and make listed under Devel



Install Cygwin on Windows (2)

- ▶ Once installed under `C:/cygwin64` you will have a Unix-like environment
- ▶ You can use it to compile and debug your code using `gcc` and `gdb`



```
and "show warranty" for details.  
This GDB was configured as "x86_64-pc-cygwin".  
Type "show configuration" for configuration details.  
For bug reporting instructions, please see:  
<http://www.gnu.org/software/gdb/bugs/>.  
Find the GDB manual and other documentation resources online at:  
<http://www.gnu.org/software/gdb/documentation/>.  
For help, type "help".  
Type "apropos word" to search for commands related to "word".  
(gdb) exit  
Undefined command: "exit". Try "help".  
(gdb) quit  
  
klipskoch@WGJSYMW1 ~  
$ gcc  
gcc: fatal error: no input files  
compilation terminated.  
  
klipskoch@WGJSYMW1 ~  
$ make  
make: *** No targets specified and no makefile found. Stop.  
  
klipskoch@WGJSYMW1 ~  
$
```

make (1)



- ▶ `make` is special utility to help programmer compiling and linking programs
- ▶ Programmers had to type in compile commands for every change in program
- ▶ With more modules more files need to be compiled
 - ▶ Possibility to write script, which handles sequence of compile commands
- ▶ Inefficient

make (2)

- ▶ Compiling takes time
- ▶ For only small change in one module, not necessary to recompile other modules
- ▶ `make`: compilations depends upon whether file has been updated since last compilation
- ▶ Also possible to specify dependencies
- ▶ Also possible to specify commands to compile (e.g., depending of suffix of source)

Makefile (1)

- ▶ A makefile has the name "Makefile"
- ▶ Makefile contains following sections:
 - ▶ Comments
 - ▶ Macros
 - ▶ Explicit rules
 - ▶ Default rules

Makefile (2)

- ▶ Comments
 - ▶ Any line that starts with a # is a comment
- ▶ Macro format
 - ▶ name = data
 - ▶ Ex: OBJ=linked_list.o use_linked_list.o
 - ▶ Can be referred to as \$(OBJ) from now on

Makefile (3)

Explicit rules

- ▶ target:source1 [source2] [source3]

command1

tabs here [command2]

[command3]



- ▶ target is the name of file to create
- ▶ File is created from source1 (and source2, ...)
- ▶ use_linked_list: use_linked_list.o linked_list.o
gcc -o use_linked_list
use_linked_list.o linked_list.o

Makefile (4)

Explicit rules



► target:

command

Commands are unconditionally executed each time make is run

- Commands may be omitted, **built-in rules** are used then to determine what to do

`use_linked_list.o: linked_list.h use_linked_list.c`

- Create `use_linked_list.o` from `linked_list.h` and `use_linked_list.c` using standard suffix rule for getting to `use_linked_list.o` from `linked_list.c`

- `$(CC) $(CFLAGS) -c file.c`



Example Makefile (1)


- ▶ Header file with `struct` definition and function prototypes
 - ▶ `header_file.h`
- ▶ Implementation file with usage of the `struct` and function definitions
 - ▶ `implementation.c`
- ▶ Main function where implemented behaviour can be used
 - ▶ `main.c`
- ▶ Makefile with different targets for different purposes
 - ▶ `Makefile.txt`

Run Makefile

- ▶ `make`
Default makefile called `Makefile` and default target `all`
- ▶ `make TargetName`
Default makefile called `Makefile` and target `TargetName`
- ▶ `make -f MyMakeFile.txt`
Makefile called `MyMakeFile.txt` and default target `all`
- ▶ `make -f MyMakeFile.txt TargetName`
Makefile called `MyMakeFile.txt` and default target `TargetName`

Example Makefile (2)

```
1 CC = gcc
2 CFLAGS = -Wall
3
4 OBJ = linked_list.o use_linked_list.o
5
6 all: use_linked_list
7
8 use_linked_list: $(OBJ)
9                 $(CC) $(CFLAGS) -o use_linked_list $(OBJ)
10
11 use_linked_list.o: linked_list.h use_linked_list.c
12
13 linked_list.o: linked_list.h linked_list.c
14
15 clean:
16     rm -f use_linked_list *.o
```



Function Pointers

- ▶ A pointer may not just point to a variable, but may also point to a function
- ▶ In the program it is assumed that the function does what it has to do and you use it in your program as if it was there
- ▶ The decision which function will actually be called is determined at run-time

Function Pointer Syntax

- ▶ `void (*foo)(int);`
 - ▶ `foo` is a pointer to a function taking one argument, an integer, and that returns `void`
- ▶ `void *(*foo)(int *);`
 - ▶ `foo` is a pointer to a function that returns a `void *` and takes an `int *` as parameter
- ▶ `int (*foo_array[2])(int);`
 - ▶ `foo_array` is an array of two pointer functions having an `int` as parameter and returning an `int`
- ▶ Easier and equivalent:
`typedef int (*foo_ptr_t)(int);`
`foo_ptr_t foo_ptr_array[2];`

Function Pointers: Simple Examples

```
1 void (*func) (void);    /* define pointer to function */
2 void a(void) { printf("func a\n"); }
3 void b(void) { printf("func b\n"); }
4
5 int main() {
6     func = &a;           // calling func() is the same as calling a()
7     func = a;            // calling func() is the same as calling a()
8     func();              // both are the same
9 }
```

One may have an array of function pointers:

```
1 int func1(void);
2 int func2(void);
3 int func3(void);
4 int (*func_arr[3])(void)
5                                     = {func1, func2, func3};
```

Another Function Pointer Example

```
1 #include <stdio.h>
2 void output(void) {
3     printf("%s\n", "Please enter a number:");
4 }
5 int sum(int a, int b) {
6     return (a + b);
7 }
8 int main() {
9     int x, y;
10    void (*fptr1)(void);
11    int (*fptr2)(int, int);
12    fptr1 = output;
13    fptr2 = sum;
14    fptr1();    // cannot see whether function or pointer
15    scanf("%d", &x);
16    (fptr1)();    // some prefer this to show it is pointer
17    (*fptr1)();    // complete syntax, same as above
18    scanf("%d", &y);
19    printf("The sum is %d.\n", fptr2(x, y));
20 }
```


Alternatives for Usage

```
1 int (*fct) (int, int);  
2 /* define pointer to a fct */  
3 int plus(int a, int b) {return a+b;}  
4 int minus(int a, int b) {return a-b;}  
5 int a=3; int b=4;  
6 fct = &plus;  
7 /* calling fct() same as calling plus() */  
8 printf("fct(a,b):%d\n", fct(a,b)); /* 7 */
```

or

```
1 printf("fct(a,b):%d\n", (*fct)(a,b)); /* 7 */  
2 fct = &minus;  
3 /* calling fct() same as calling minus() */  
4 printf("fct(a,b):%d\n", fct(a,b)); /* -1 */
```