

C++ For C Coders

Data Structures
C++ for C Coders

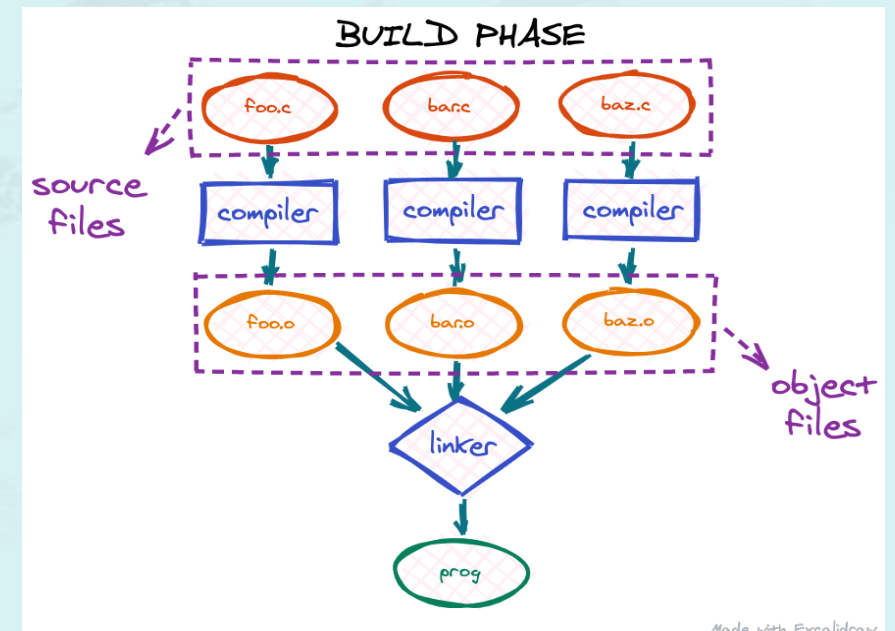
한 동 대 학 교 김영섭 교수
idebtor@gmail.com

Introduction to the GNU C preprocessor
Header Files
Macros
Conditionals

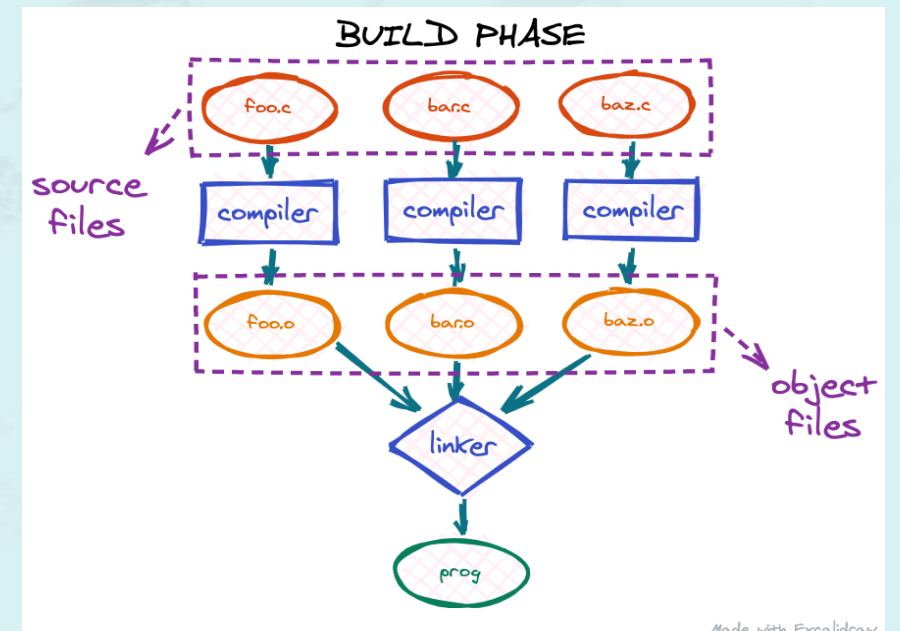
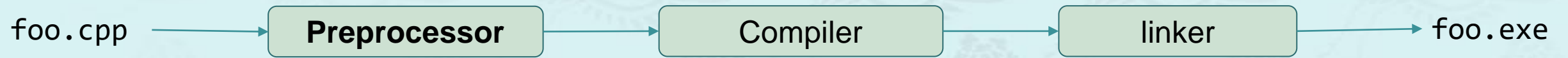
NMN, DRY, KISS, NSE

Introduction to the GNU C preprocessor

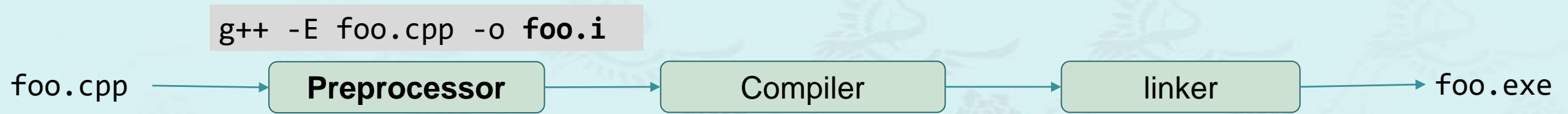
foo.cpp → `g++ -std=c++11 foo.cpp -o foo` → foo.exe



Introduction to the GNU C preprocessor



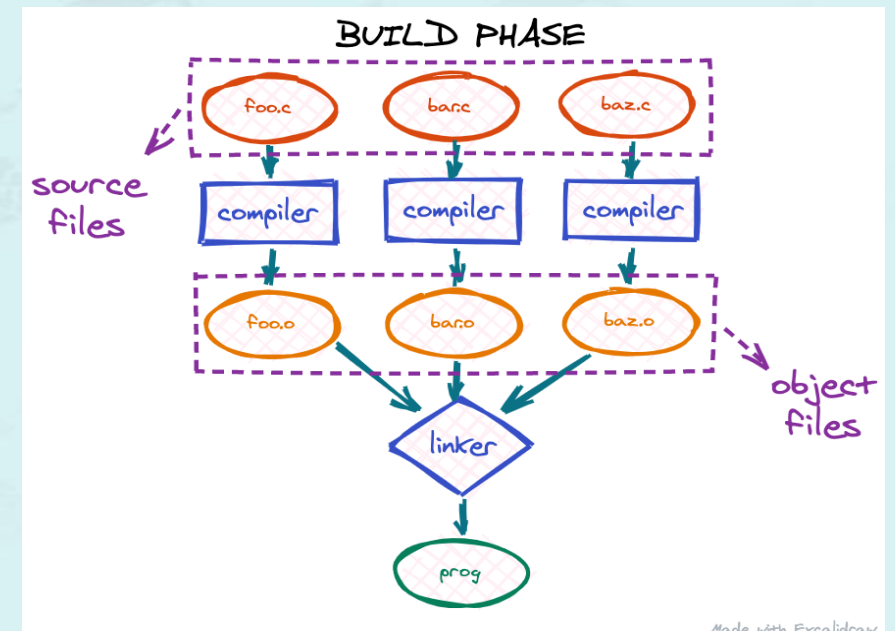
Introduction to the GNU C preprocessor



foo.i

- Header files are included.
- Macros are replaced.
- Comments are removed.

```
#define
#include
#ifdef          #endif
#if defined     #endif
#ifndef        #endif
#if            #endif
#if           #else  #endif
```



Introduction to the GNU C preprocessor

```
g++ -c foo.cpp -o foo.o
```

```
g++ -E foo.cpp -o foo.i
```

foo.cpp

Preprocessor

Compiler

linker

foo.exe

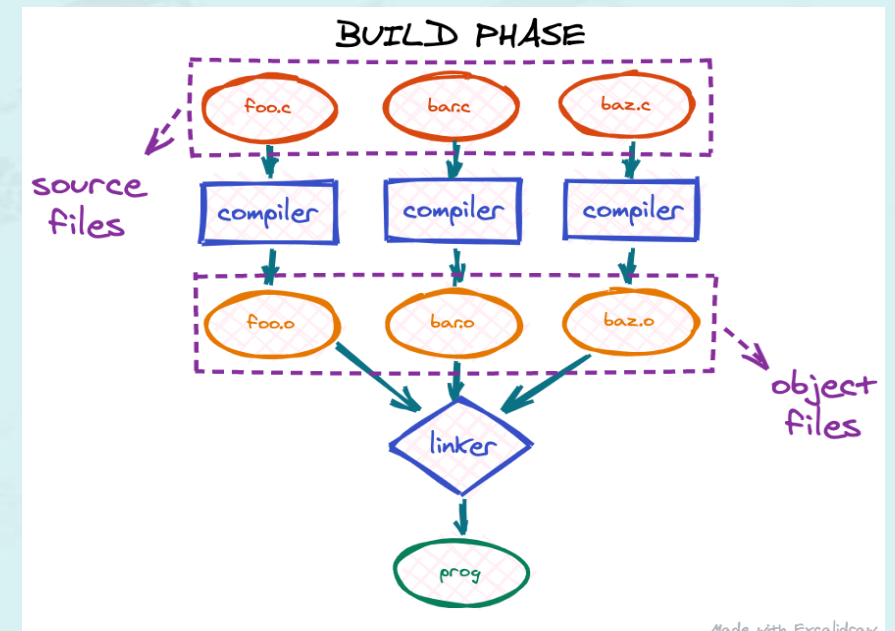
foo.i

- Header files are included.
- Macros are replaced.
- Comments are removed.

foo.o

- object file

```
#define
#include
#ifdef          #endif
#if defined     #endif
#ifndef         #endif
#if            #endif
#if            #endif
#if            #else    #endif
```



Introduction to the GNU C preprocessor

```
g++ -c foo.cpp -o foo.o
```

```
g++ -E foo.cpp -o foo.i
```

```
g++ foo.o -o foo.exe
```

foo.cpp

Preprocessor

Compiler

linker

foo.exe

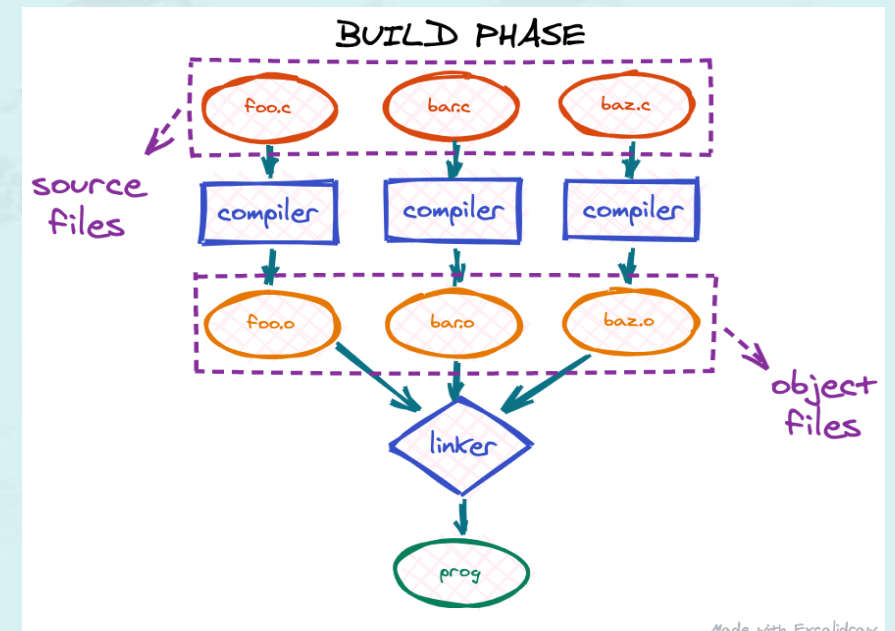
foo.i

- Header files are included.
- Macros are replaced.
- Comments are removed.

foo.o

- object file

```
#define
#include
#ifdef          #endif
#if defined     #endif
#ifndef         #endif
#if            #endif
#if            #endif
#if            #else    #endif
```



Four main macro directives

1. Macros
2. File Inclusion
3. Conditional Compilation
4. Other directives
5. Predefined macros
6. Ref: <https://gcc.gnu.org/onlinedocs/cpp/>

1. Macros #define

Macros are pieces of code in a program that is given some name.

- #define - substitutes a preprocessor macro.

```
#include <iostream>

// macro definition
#define PI 3.141592

int main() {
    double radius = 5;
    double area = PI * radius * radius;
    std::cout << "area = " << area << "\n";

    return 0;
}
```


2. Macros with Arguments

Macros are pieces of code in a program that is given some name.

- #define - substitutes a preprocessor macro.

```
#include <iostream>

// macro with parameter
#define SQUARE(a) ((a) * (a))
int main() {
    int squared;
    int x = 100;
    squared = SQUARE(x);
    std::cout << "squared = " << squared;
    return 0;
}
```

```
#include <iostream>

// macro with parameter
#define MAX(x, y) ((x) > (y) ? (x) : (y))
int main() {
    std::cout << "Max = " << MAX(10, 20);
    return 0;
}
```

3. File Inclusion #include

Macros are pieces of code in a program that is given some name.

- #include - inserts a particular header from another file.
- There are two types of files that can be included by the user in the program: Standard files and User-defined files.

```
#include <file_name>      // standard files  
#include "file_name"
```

```
#include <stdio.h>  
#include "myheader.h"
```

- The *filepath* is given to specify the directory.
- The contents of the header file is directly copy-pasted to the *sourcefile*.

```
g++ sourcefile -I filepath
```

4. Conditional Compilation

Controls the execution of the surrounded code.

- The 3 reasons it is used:
 - For different Operating Systems(Linux, MacOS, etc.)
 - To compile into different versions, using the same source file.
 - To refer as a comment.

Conditional Compilation directives:

- `#undef` – undefines a preprocessor macro.
- `#ifdef` – returns true if this macro is defined.
- `#ifndef` – returns true if this macro is not defined.
- `#if` – tests if a compile time condition is true.
- `#else` – the alternative for `#if`.
- `#elif` – `#else` and `#if` in one statement.
- `#endif` - ends preprocessor conditional.
- `#error` – prints error message on stderr.
- `#pragma` – issues special command to the compiler. compiler specific

4. Conditional Compilation Examples

```
#undef  FILE_SIZE
#define FILE_SIZE 10
```

```
#ifndef MESSAGE
    #define MESSAGE "Hello!"
#endif
```

```
#ifdef DEBUG
    // Your debugging statements here
#endif
```

This is useful if you pass the **-DDEBUG** flag to the gcc compiler at the time of compilation. This will define **DEBUG**, so you can turn debugging on and off on the fly during compilation.

```
#ifdef DEBUG
    #define DPRINT(func) func;
#else
    #define DPRINT(func) ;
#endif
```

Now, can you interpret what this macro does?



4. Conditional Compilation: #error

- Example, the directives

```
#define BUFFER_SIZE 255  
  
#if BUFFER_SIZE < 256  
#error "BUFFER_SIZE is too small."  
#endif
```

- generate the error message

```
BUFFER_SIZE is too small.
```

4. Example: Quicksort

```
#include <iostream>
using namespace std;

#ifdef DEBUG
#define DPRINT(func) func;
#else
#define DPRINT(func) ;
#endif

// This function takes last element as pivot,
// places the pivot element at its
int partition(int list[], int lo, int hi) {
    DPRINT(cout << "partition pivot:" << list[hi] << endl);
    int pivot = list[hi];          // pivot value
    int i = (lo - 1);              // smaller element index
    for (int j = lo; j <= hi - 1; j++) {
        if (list[j] <= pivot) {    // left side <= pivot
            i++;                  // smaller element index
            swap(list[j], list[i]); // swap two
        }
    }
    swap(list[hi], list[i + 1]);
    return (i + 1); // returns the new pivot index.
}
```

```
void quicksort(int *list, int lo, int hi) {
    if (lo < hi) {
        int pi = partition(list, lo, hi); // pi: pivot index
        DPRINT(cout << " l pivot(" << pi << ")=" << list[pi] << "\n");
        quicksort(list, lo, pi - 1);
        DPRINT(cout << " r pivot(" << pi << ")=" << list[pi] << "\n");
        quicksort(list, pi + 1, hi);
    }
}

void quicksort(int *a, int n) {
    quicksort(a, 0, n - 1);
}

#ifdef 1
int main() {
    int list[] = { 54, 26, 93, 17, 77, 31, 44, 55, 20 };
    cout << "UNSORTED: \n";
    for (auto x: list) cout << x << " "; cout << endl;

    quicksort(list, sizeof(list) / sizeof(list[0]));
    cout << "QUICK SORTED: \n";
    for (auto x: list) cout << x << " "; cout << endl;
}
#endif
```

4. Example: Quicksort

```
#include <iostream>
using namespace std;

#ifdef DEBUG
#define DPRINT(func) func;
#else
#define DPRINT(func) ;
#endif

// This function takes last
// places the pivot element
int partition(int list[], int lo, int hi) {
    DPRINT(cout << "partition\n");
    int pivot = list[hi];
    int i = (lo - 1);
    for (int j = lo; j <= hi - 1; j++) {
        if (list[j] <= pivot) { // left side <= pivot
            i++;                // smaller element index
            swap(list[j], list[i]); // swap two
        }
    }
    swap(list[hi], list[i + 1]);
    return (i + 1); // returns the new pivot index.
}
```

```
PS C:\GitHub\nowicx\labs\lab6> g++ quicksort.cpp -o quicksort
PS C:\GitHub\nowicx\labs\lab6> ./quicksort
UNSORTED:
54 26 93 17 77 31 44 55 20
QUICK SORTED:
17 20 26 31 44 54 55 77 93
PS C:\GitHub\nowicx\labs\lab6> 
```

```
void quicksort(int *list, int lo, int hi) {
    if (lo < hi) {
        int pi = partition(list, lo, hi); // pi: pivot index
        DPRINT(cout << " l pivot(" << pi << ")=" << list[pi] << "\n");
        quicksort(list, lo, pi - 1);
        DPRINT(cout << " r pivot(" << pi << ")=" << list[pi] << "\n");
        quicksort(list, pi + 1, hi);
    }
}
```

```
int main() {
    int list[] = { 54, 26, 93, 17, 77, 31, 44, 55, 20 };
    cout << "UNSORTED: \n";
    for (auto x: list) cout << x << " "; cout << endl;

    quicksort(list, sizeof(list) / sizeof(list[0]));
    cout << "QUICK SORTED: \n";
    for (auto x: list) cout << x << " "; cout << endl;
}
#endif
```

4. Example: Quicksort

```
#include <iostream>
using namespace std;

#ifdef DEBUG
#define DPRINT(func) func;
#else
#define DPRINT(func) ;
#endif

// This function takes last element
// places the pivot element at its
int partition(int list[], int lo, int hi) {
    DPRINT(cout << "partition pivot:" << list[hi] << "\n");
    int pivot = list[hi]; // pivot
    int i = (lo - 1); // small
    for (int j = lo; j <= hi - 1; j++) {
        if (list[j] <= pivot) { // left
            i++; // small
            swap(list[j], list[i]); // swap
        }
    }
    swap(list[hi], list[i + 1]);
    return (i + 1); // returns the new pivot index.
}
```

```
void quicksort(int *list, int lo, int hi) {
    if (lo < hi) {
        int pi = partition(list, lo, hi); // pi: pivot index
        DPRINT(cout << " l pivot(" << pi << ")=" << list[pi] << "\n");
        quicksort(list, lo, pi - 1);
        quicksort(list, pi + 1, hi);
    }
}

int main() {
    int n;
    int list[n];
    for (int i = 0; i < n; i++) {
        list[i] = 54, 26, 93, 17, 77, 31, 44, 55, 20;
    }
    cout << "UNSORTED: \n";
    for (auto x: list) cout << x << " "; cout << endl;
    quicksort(list, 0, list.size() - 1);
    cout << "QUICK SORTED: \n";
    for (auto x: list) cout << x << " "; cout << endl;
}

#endif
```

```
PS C:\GitHub\nowicx\labs\lab6> g++ -DDEBUG quicksort.cpp -o quicksort
PS C:\GitHub\nowicx\labs\lab6> ./quicksort
UNSORTED:
54 26 93 17 77 31 44 55 20
partition pivot:20
l pivot(1)=20
r pivot(1)=20
partition pivot:26
l pivot(2)=26
r pivot(2)=26
partition pivot:93
l pivot(8)=93
partition pivot:55
l pivot(6)=55
partition pivot:44
l pivot(4)=44
r pivot(4)=44
r pivot(6)=55
r pivot(8)=93
QUICK SORTED:
17 20 26 31 44 54 55 77 93
PS C:\GitHub\nowicx\labs\lab6>
```


5. Predefined Macros

Macro	Value
__DATE__	A string containing the current date.
__FILE__	A string containing the file name.
__LINE__	An integer representing the current line number.
__STDC__	If follows ANSI standard C, then the value is a nonzero integer.
__TIME__	A string containing the current time.

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

```
int main() {  
    printf("File :%s\n", __FILE__ );  
    printf("Date :%s\n", __DATE__ );  
    printf("Time :%s\n", __TIME__ );  
    printf("Line :%d\n", __LINE__ );  
    printf("ANSI :%d\n", __STDC__ );  
}
```

```
File :test.cpp  
Date :Mar 5 2023  
Time :22:46:24  
Line :7  
ANSI :1
```

6. Header Guards

Example(rand.h):

```
#ifndef RAND_H
#define RAND_H

unsigned long rand_extended();

void randomize(int list[], int size);
int *randomize_insideout(int* list, int size);
void randomize_naive(int list[], int size);

#endif
```

Avoiding Macros in C++

- In C++, you should generally **avoid macros** when possible.
- **Inline functions** should also get rid of the need for macros for efficiency reasons.
- **Use const** to declare typed constants rather than #define to create untyped (and therefore less safe) constants.

In-house Coding Principles

- NMN – No Magic Number
- DRY – Do not Repeat Yourself
- NSE – No Side Effect
- KISS – Keep It Simple, Stupid!

NMN - No Magic Number

Example

```
#include <iostream>
using namespace std;

int main(int argc, char **argv) {
    ...
}
```

Homework !

NSE – No Side Effect!

Example

```
#include <iostream>
using namespace std;

int add(int num1, int num2);

int main(int argc, char **argv) {
    int num1 = 5;
    int num2 = 5;

    int sum = add(num1, num2);
    printf("sum is: %d\n", &sum);
}

int add(int num1, int num2) {
    int sum = num1 + num2;

    printf("sum is: %d\n", sum);

    return sum;
}
```

Result:

\$ sum is: 10

\$ sum is: 10