

BUCKNELL UNIVERSITY
Computer Science

CSCI 315 Operating Systems Design

Overview of C Programming

Notice: This set of slides is based on the notes by Professor Perrone of Bucknell and Professor Phil Kearns of William & Mary.

What is C?

- C is a high level programming language used mostly for systems programming including operating systems.
- C was created between 1969 and 1973 by Dennis Ritchie of AT&T Bell Lab when designing and implementing the UNIX operating system
- C is the used to develop Linux as well
- Many modern programming languages borrowed ideas and features from C, including C#, D, Go, Rust, Java, JavaScript, Limbo, LPC, Objective-C, Perl, PHP, and Python
 - [http://en.wikipedia.org/wiki/C_\(programming_language\)](http://en.wikipedia.org/wiki/C_(programming_language))

A Short C Program

- *hello.c*:

Similar to “import” in Java or Python

```
#include <stdio.h>
int main(int argc, char* argv[]) {
    printf("Hello World!\n");
    return 0;
}
```

*Identical to Java,
similar to Python
function*

- Compile and execute a C program:

```
%gcc -o hello hello.c
%./hello
```

Why C?

- In contrast to Python or Java:
 - C has direct access to operating systems resources, libraries.
 - C is efficient (light-weight)
 - C is extensively used in embedded systems and operating systems
 - C requires minimal run-time support

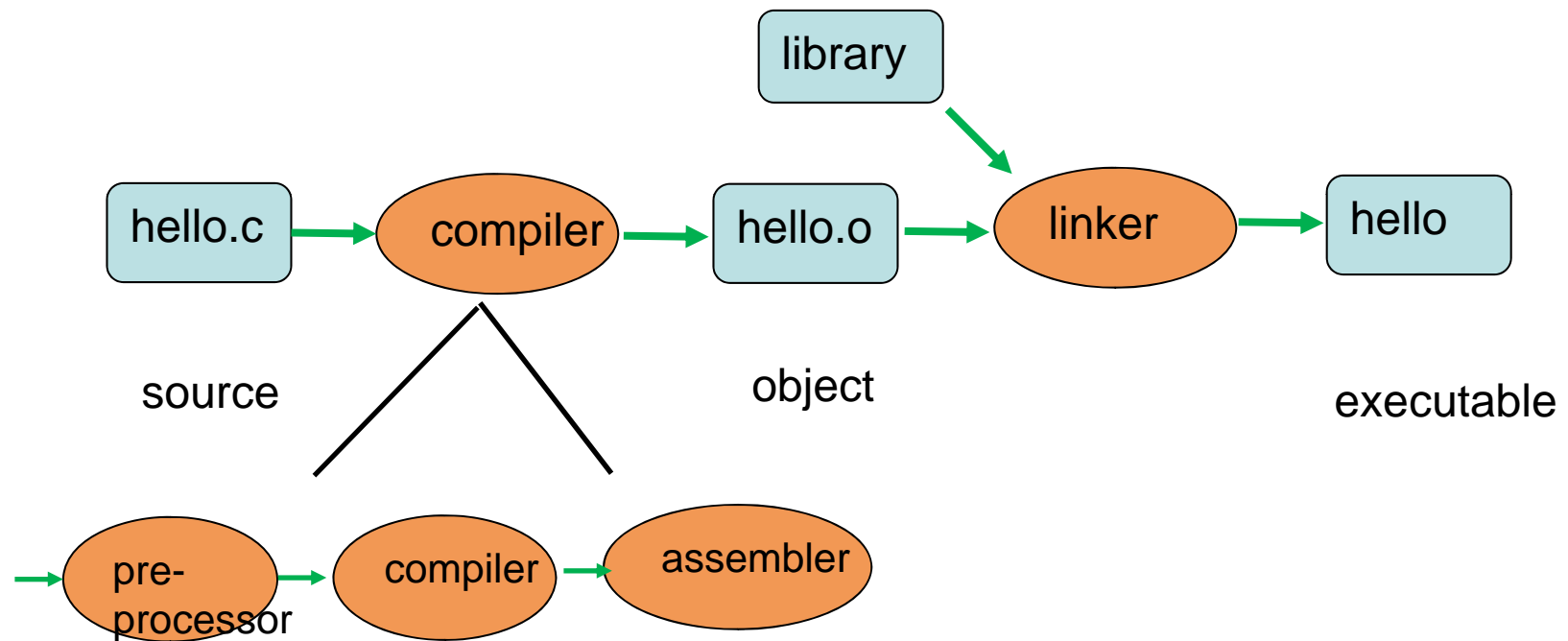
Pitfalls of C

- We will see many pitfalls of C, two most critical ones:
 - Pointers and addresses
 - Cryptic error messages (though they have been much improved over the years)

Workflow of C Programs



Workflow of C Programs



Basic Data Types

```
#include <stdio.h>
int main(int argc, char*argv[]) {

    int i = 7;
    float x = 2.71828;
    double y = 3.1415926;
    char c = 'w';

    printf ("%d, %c, %f, %lf\n", i, c, x, y);

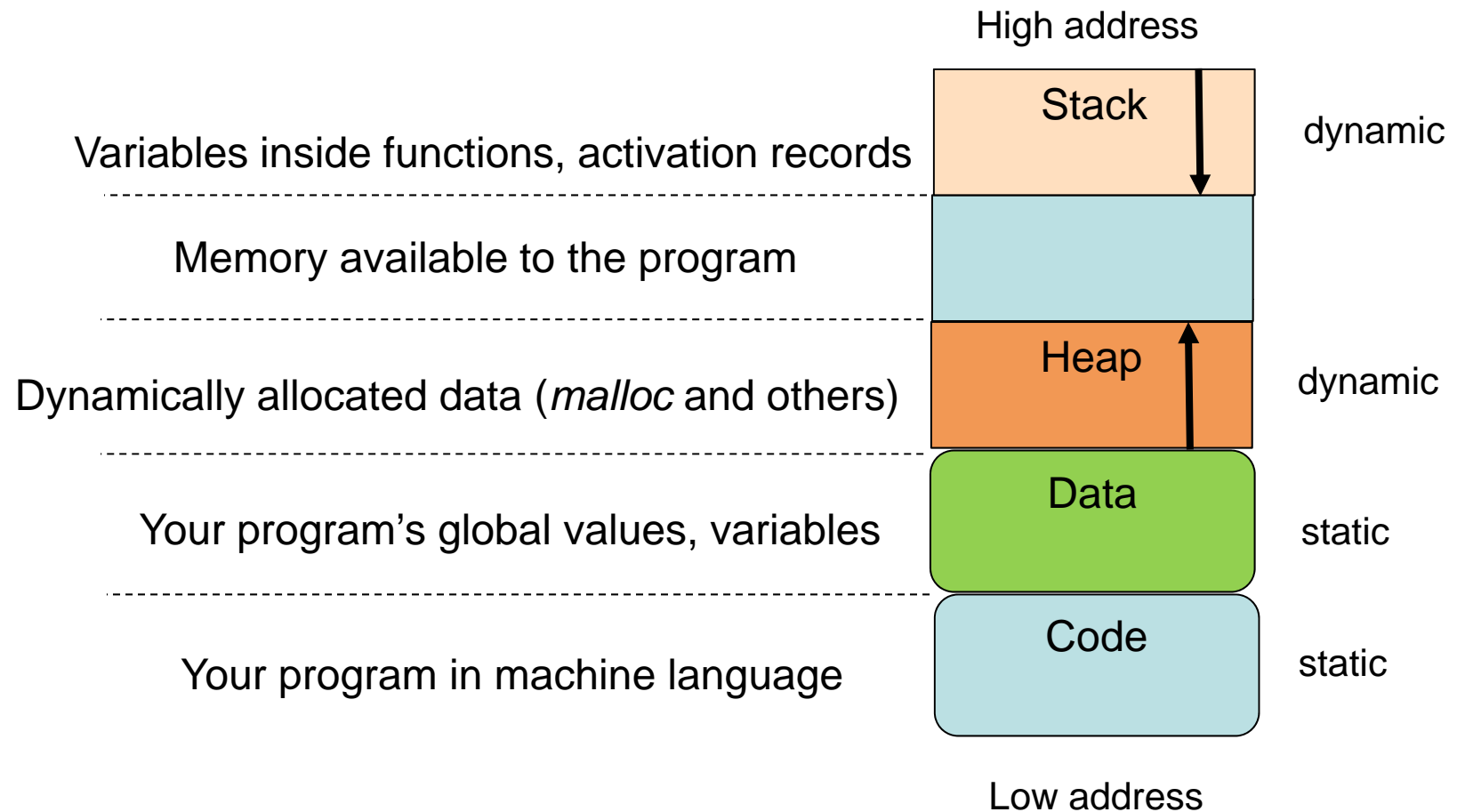
    return 0;
}
```


Scope

```
{  
    int i = 999;  
    int j = 666;  
    printf("i = %d, j = %d\n", i, j);  
    {  
        int i = 123;  
        int j = i*i;  
        printf("i = %d, j = %d\n", i, j);  
    }  
    printf("i = %d, j = %d\n", i, j);  
}
```

```
i = 999, j = 666  
i = 123, j = 15129  
i = 999, j = 666
```

Program's View of Memory



Repetitions

```
int sum = 0;  
int k;  
for (k = 0; k < 100; k ++)  
    sum += k;
```

```
int sum = 0;  
int k = 0;  
do {  
    sum += k;  
    k ++;  
} while (k < 100);
```

```
int sum = 0;  
int k = 0;  
while (k < 100) {  
    sum += k;  
    k ++;  
}
```

Selections

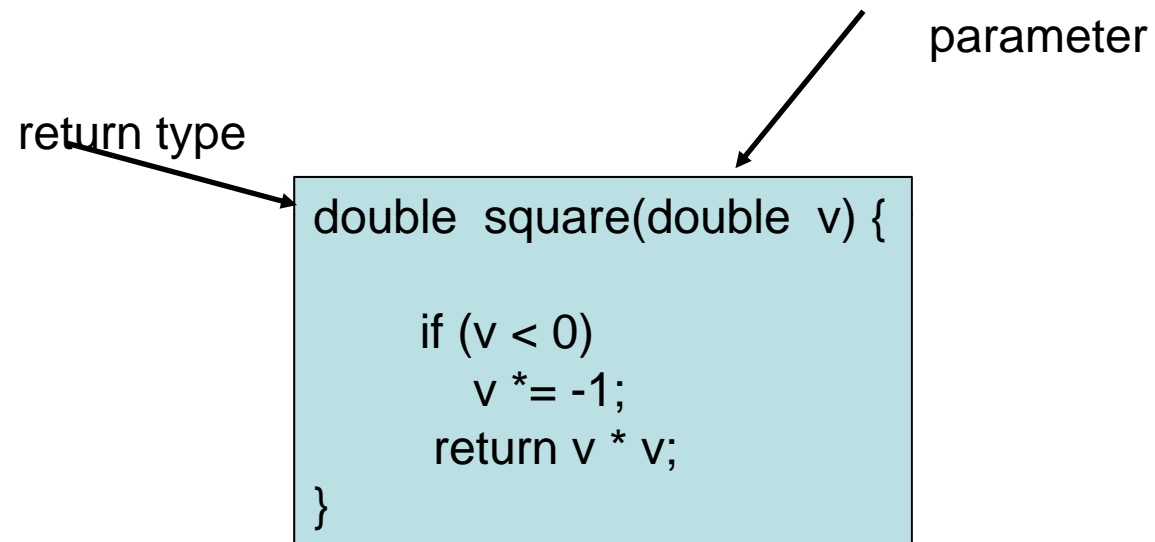
```
if (k < 100) {  
    sum += k;  
}
```

```
if (k < 100) {  
    sum += k;  
} else {  
    sum += 1;  
}
```

```
if (k < 100 && sum < 1000) {  
    sum += k;  
}
```

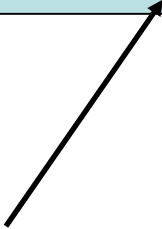
```
switch (score) {  
    case 0:  
        grade = 'A';  
        break;  
    case 1:  
        grade = 'B';  
        break;  
    default:  
        grade = 'C';  
}
```

Functions



Arrays

```
double values[10];  
  
values[0] = 1;  
values[1] = 3;  
...  
values[10] = 14;
```



Out of range! C compiler doesn't check it. You have to!

Structures

Structure is a type definition

```
struct employee {  
    char * name;  
    int    id;  
    double wage;  
};
```

Define a variable of the type *employee*

```
struct employee boss;  
struct employee programmers[20];
```

Pass structure as parameter(s) and access its fields

```
void print_employee(struct employee person) {  
    printf("name : %s\n", person.name);  
    printf("id %d and salary %12.2f\n", person.id, person.wage);  
}
```

Pointers

Pointers are just addresses to a variable

1. Declarations:

```
char * s;  
int * p;  
int k;  
struct employee * e;
```

2. Allocate memory

```
struct employee e = (struct employee *)malloc(sizeof(struct employee));  
e->name = (char*)malloc(30);  
s = (char *)malloc(20);
```

3. Some operations

```
e->id = k;  
strcpy(e->name, "Jane Doe");  
p = &k;  
printf("name : %s\n", e->name);  
p = &(e->id);
```


Pointer and array

A pointer is simply an address for a variable;
So is the name of an array;

```
char * s;  
s = (char*)malloc(20);  
s[0] = 'a';  
s[1] = 'b';  
s[19] = '\0'; // or s[19] = 0;
```

$\&(s[i]) == s+i$



```
struct employee * e = (struct employee *)malloc(3 * sizeof(struct employee));  
e[2].id = 4; strcpy(e[1].name, "Alice"); strcpy(e[0].name, "Bob");  
printf("e[2].id == %d --- e[0].name == %s\n", e[2].id, e[0].name);
```

```
e[2].id == 4 --- e[0].name == Bob
```



Printed result

Multi-file programs

main.c

```
void print_array(int[], int); // prototype

int main(int argc, char*argv[]) {

    int v[] = {4, 6, 2, 1};
    print_array(v, 4);

    return 0;
}
```

support.c

```
#include <stdio.h>
void print_array(int v[], int c) {
    int k = 0;
    for (k = 0; k < c; k++)
        printf("v[%d] == %d\n",k,v[k]);
}
```

Linux command to compile and run the program *main*

```
%gcc -c main.c support.c
%gcc -o main main.o support.o
%./main
```

Printed result →

```
v[0] == 4
v[1] == 6
v[2] == 2
v[3] == 1
```

Input in C

Reading information into variables in C can be tricky.
The general concept is to read information into the address of a variable.

Assume we have defined:

```
int k;  
char s[32];  
char c;  
FILE * f = fopen("test.txt", "r");
```

Reading from keyboard (stdin):

```
scanf("%d", &k);  
scanf("%s", s);  
scanf("%c", &c);
```

Reading from a file:

```
fscanf(f, "%d", &k);  
fscanf(f, "%s", s);  
fscanf(f, "%c", &c);
```

s can't contain white spaces

Reading strings

`scanf("%s", s)` will stop at any white space, consecutive calls to `scanf()` will skip the white space chars.

For example, if the input is "hello world!", the above statement will only read "hello" into `s`. Next call to `scanf()` will read "world!" into a variable

To read text containing white spaces (' ', tab, ret), use `gets()` from keyboard input, or `fgets()` from a file. `fgets()` stops at the first occurrence of the newline char.

Caution: `scanf()` will leave the newline char in the buffer, while `fgets()` will store the newline char with the read buffer.

What will these two lines do for input "hello\nworld?"?

```
fscanf(f, "%s", s1);  
fgets(f, "%s", s2).
```

```
s1 == "hello"  
s2 == "\n"
```

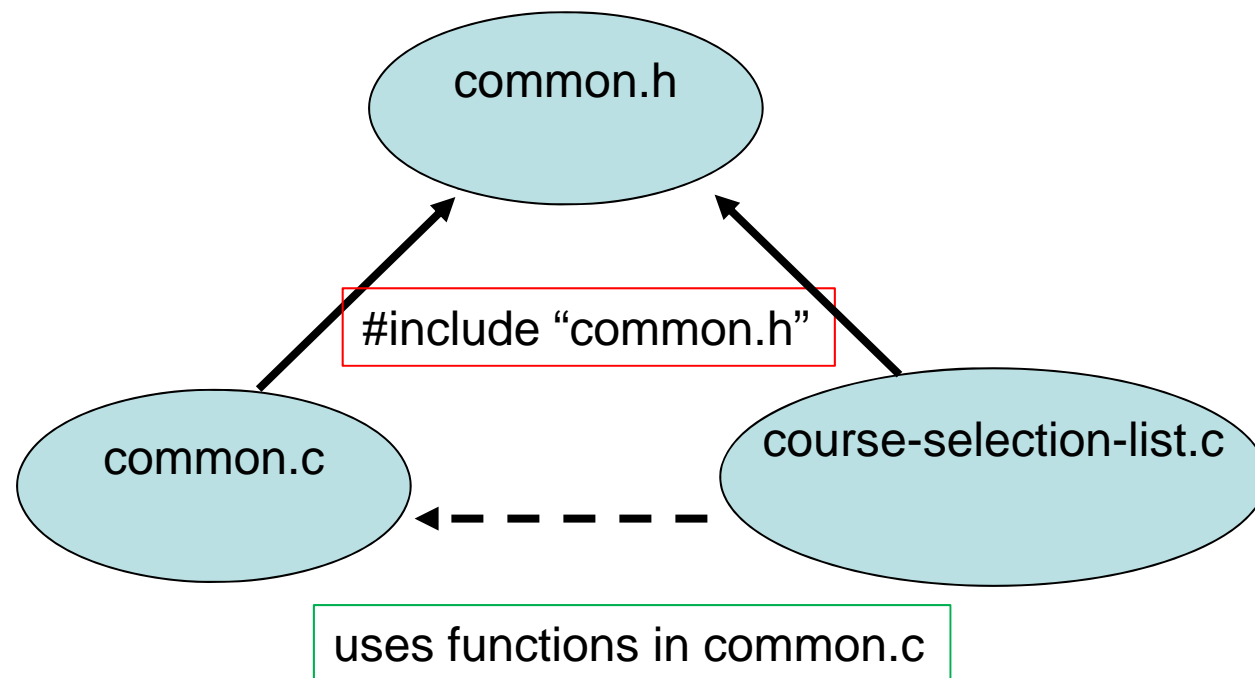
An application problem

- Read a file containing a student course selection information
- Print the contents
- Perform search as needed

Major components

- Read text file
- Repetitions
- String comparison
- Function calls
- Define and use structures
- Arrays and linked lists
- See <<http://www.eg.bucknell.edu/~cs315/2013-fall/code-examples/c-intro/>>

Structure of the program



Flow of program execution

