

C basics

Lecture 02.01

C is a language for fast small programs

- It is used where speed, space, and portability are important
- Where?
 - Most operating systems are written in C.
 - Most other computer languages are written in C
 - Most games are written in C
- It creates code which is much closer to the language that machine can understand

What language
machines can
understand?

To write a working C program you need

- Operating system (we develop for [Linux](#))
- Text editor (choose your favorite, but vi is always available)
- Compiler (we use [gcc](#))

- All this is available on the src-code server

The way C works

Text file rocks.c

```
#include <stdio.h>
int main() {
    puts("C Rocks!");
    return 0;
}
```

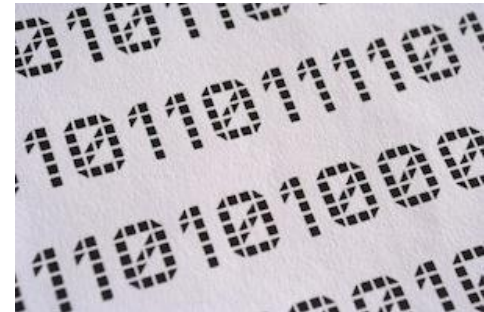
Human-readable code

Compiler

```
gcc rocks.c -o rocks
```

*Run through a compiler,
which translates C code
into machine code*

Machine code



*Executable **rocks** is a
program that computer
can understand*

Anatomy of a complete C program

```
/*  
* Program to calculate the number of cards in the shoe.  
* This code is released under the Vegas Public License. */
```

Comments: explain
what program does

```
#include <stdio.h>  
  
int main() {  
    int decks;  
    puts("Enter a number of decks");  
    scanf("%i", &decks);  
    if (decks < 1) {  
        puts("That is not a valid number of decks");  
        return 1;  
    }  
    printf("There are %i cards\n", (decks * 52));  
    return 0;  
}
```

Anatomy of a complete C program

```
/*  
* Program to calculate the number of cards in the shoe.  
* This code is released under the Vegas Public License. */
```

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

```
int main() {  
    int decks;  
    puts("Enter a number of decks");  
    scanf("%i", &decks);  
    if (decks < 1) {  
        puts("That is not a valid number of decks");  
        return 1;  
    }  
    printf("There are %i cards\n", (decks * 52));  
    return 0;  
}
```

Here you tell the compiler to include code from other libraries.

The *stdio* library contains code for reading and writing data from and to the terminal.

Anatomy of a complete C program

```
/*  
 * Program to calculate the number of cards in the shoe.  
 * This code is released under the Vegas Public License. */  
#include <stdio.h>
```

```
int main() {  
    int decks;  
    puts("Enter a number of decks");  
    scanf("%i", &decks);  
    if (decks < 1) {  
        puts("That is not a valid number of decks");  
        return 1;  
    }  
    printf("There are %i cards\n", (decks * 52));  
    return 0;  
}
```

C code consists of
functions.

The *main* function is a starting point of any program.

It is expected to return an integer:

- 0 on success
- anything else on error

Compile && run

```
gcc rocks.c -o rocks && ./rocks
```

- The GNU Compiler Collection (gcc):
 - Compiles for many operating systems
 - Produces machine code for many hardware configurations
 - Compiles lots of languages other than C
 - Completely free

C syntax

- Compact
- Simple
- Modular

- Influenced many other languages, including the most popular Java and JavaScript

Recap: Conditionals

```
if (fuel > 3)
    puts("It's OK. You can drive downtown.\n");
else
    if (money > 10)
        puts("You should buy some gas.\n");
    else
        puts("Sorry. Better stay at home.\n");
```

Recap: Conditionals

if (fuel > 3)

```
puts("It's OK. You can drive downtown.\n");  
drive("downtown");
```

else

if (money > 10)

```
puts("You should buy some gas.\n");  
buy_gas(money);
```

else

```
puts("Sorry. Better stay at home.\n")
```

Does the
program
compile?

Recap: Conditionals

```
if (fuel > 3)
```

```
    puts("It's OK. You can drive downtown.\n");  
    drive("downtown");
```

```
if (fuel == 0)
```

```
    if (money > 10)
```

```
        puts("You should buy some gas.\n");  
        buy_gas(money);
```

```
    if (money == 0)
```

```
        puts("Sorry. Better stay at home.\n");
```

Does the
program work
as intended?

Recap: version of if-else - switch

```
char grade = 'D';  
switch (grade) {  
    case 'F' :  
        printf("Better try again.\n" );  
    case 'D' :  
        printf("You passed.\n" );  
    case 'C' :  
    case 'B' :  
        printf("Well done.\n" );  
    case 'A' :  
        printf("Excellent!\n" );  
    default :  
        printf("Invalid grade\n" );  
}
```

Does the program
work as intended?

What is printed
here?

Recap: loops – while vs. do while

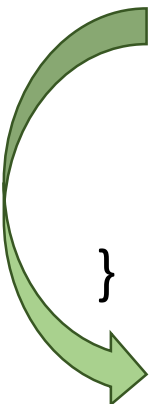
```
while (instructor_is_sick) {  
    skip_class();  
}
```

What is the difference?

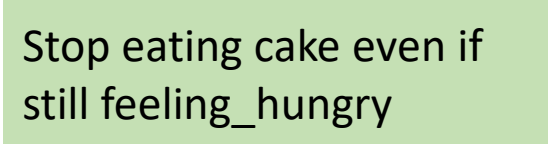
```
do {  
    skip_class();  
} while (instructor_is_sick);
```

Recap: break

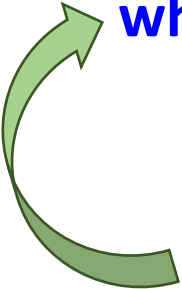
```
while (feeling_hungry) {  
    eat_cake();  
    if (feeling_queasy) {  
        /* Break out of the while loop */  
        break;  
    }  
}
```



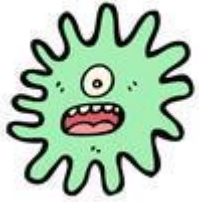
Stop eating cake even if
still feeling_hungry



Recap: continue



```
while (feeling_hungry) {  
    if (not_lunch_yet) {  
        /* Go back to the loop condition */  
        continue;  
    }  
    eat_cake();  
}
```

The story of breaks

Scary bug stories

- Breaks don't break if statements!
- AT&T crash 1990:
 - A developer used break to break out of if statement.
 - Result: the program skipped the entire section of code and interrupted phone services of 70 million people for over 9 hours

PRINTF family


- `printf`
 - Prints formatted output to standard output
- `fprintf`
 - Prints formatted output to a file
- `sprintf`
 - Prints formatted output to a string

printf

```
char * name = "Bob";
```

```
int age = 5;
```

Substitutes variable parts
with values in variables
name and *age*



```
printf ("My name is %s, I am %d years old", name, age);
```



Prints constant
parts unchanged

```
My name is Bob, I am 5 years old
```

fprintf

```
FILE * outputFP = open_file_for_writing (file_name);
```

```
fprintf (outputFP,  
        "My name is %s, I am %d years old",  
        name, age);
```

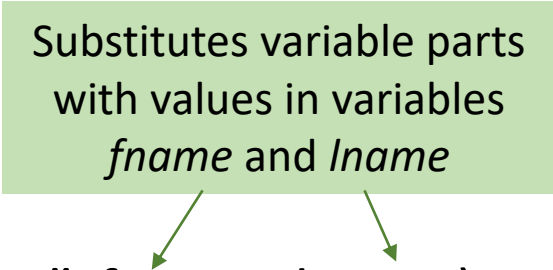
In file <file_name>

```
My name is Bob, I am 5 years old
```

sprintf

```
char full_name [80];  
char * fname = "Ben";  
char * lname = "Cook";
```

Substitutes variable parts
with values in variables
fname and *lname*



```
sprintf (full_name, "%s %s", fname, lname);
```

```
printf ("Full name is %s", full_name);
```

Full name is Ben Cook

sprintf: the C way to concatenate strings and numbers

```
char file_name [80];
```

```
char * file_prefix = "Output";
```

```
int file_number= 1;
```

```
sprintf (file_name, "%s_%d", file_prefix, file_number);
```

```
printf ("%s", file_name);
```

```
Output_1
```

SCANF family

- scanf
 - Scans standard input and fills values of variables
- fscanf
 - Scans formatted file and fills values of variables
- sscanf
 - Scans formatted string and extracts from it values to fill variables

Changing value of a function argument inside the function

```
void go_south_east (int lat, int lon) {  
    lat = lat -1;  
    lon = lon +1;  
}
```

```
int lat = 35;
```

```
int lon = -65;
```

```
go_south_east (lat, lon);
```

```
printf ("Avast! Now at %d, %d", lat, long);
```

What is printed?

Pass an address of a variable to change its value inside function

```
void go_south_east (int* lat, int* lon) {  
    *lat = *lat -1;  
    *lon = *lon +1;  
}
```

```
int lat = 35;
```

```
int lon = -65;
```

```
go_south_east (&lat, &lon);
```

```
printf ("Avast! Now at %d, %d", lat, long);
```

What is printed now?

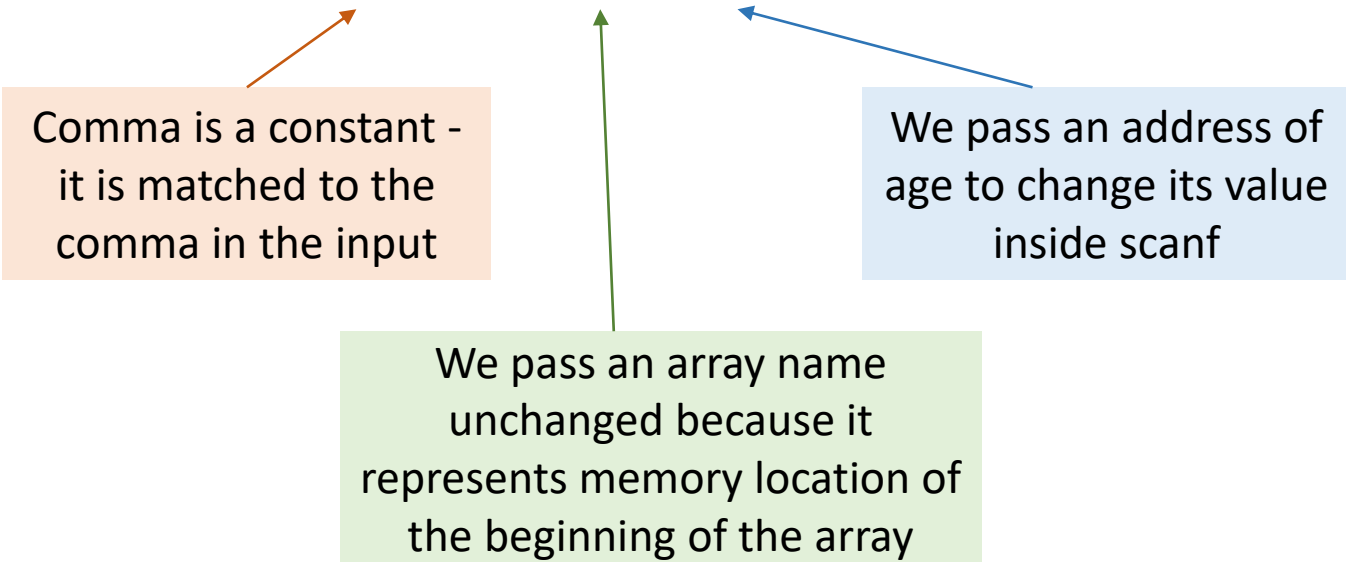
scanf

```
int age;
```

```
char *name;
```

```
printf ("Enter your name and age separated by comma: ");
```

```
scanf ("%s,%d", name, &age);
```



Comma is a constant -
it is matched to the
comma in the input

The diagram features three callout boxes with arrows pointing to the scanf function call. An orange box on the left points to the comma in the format string. A light blue box on the right points to the &age argument. A light green box at the bottom points to the name argument.

We pass an address of
age to change its value
inside scanf

We pass an array name
unchanged because it
represents memory location of
the beginning of the array

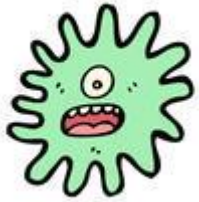
scanf returns

the number of items assigned into variables

Since assignment into variables stops when we have an invalid input for a certain format specifier, this can tell you if you've input all your data correctly:

```
int res = scanf ("%s,%d", name, &age);
```

If input is correct, then res = 2



Story of char limits

Scary bug stories

```
char food[5];  
printf("Enter favorite food: ");  
scanf("%s", food);  
printf("Favorite food: %s\n", food);
```

```
> ./food  
Enter favorite food: liver-tangerine-raccoon-toffee  
Favorite food: liver-tangerine-raccoon-toffee  
Segmentation fault: 11  
>
```

Setting limits to char arrays

- Carefully put a limit on the number of characters that scanf() will read into a string. The rest will be ignored

```
char food[5]; // filled with: {'\0', '\0', '\0', '\0', '\0'}
```

```
printf("Enter favorite food: ");
```

```
scanf("%4s", food);
```

```
printf("Favorite food: %s\n", food);
```

```
scanf("%39s", name);
```

```
scanf("%2s", card_name);
```

scanf scans strings until the first whitespace

`%s`

- Stops on the first whitespace character reached, or at the specified field width (e.g. "%10s"), whichever comes first
- To read strings with whitespaces we need to use:

`%[`

- It allows you to specify a set of valid characters
- Conversion stops when a character that is not in the set is matched

%[modifier examples

%[0-9]

- match all numbers zero **through** nine. Stop when anything else is matched

%[AD-G34]

- match A, D through G, **3, or 4**

%[^A-C]

- match all characters that are **NOT** A through C

% [^\n]

- match all the characters until the end of the line

%79 [^\n]

- match at most 79 characters or until the end of the line whichever comes first

Enough knowledge for A 1