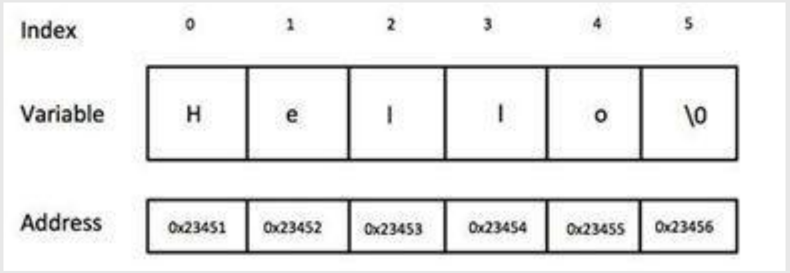
# Comp201 Spring26

## Strings in C

## C-Strings

- 1-D array of characters
- Terminated by null or \0
- Initializing a String
  - char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
  - char greeting[] = "Hello";



| Index | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Variable | H | e | l | l | o | \0 |
| Address | 0x23451 | 0x23452 | 0x23453 | 0x23454 | 0x23455 | 0x23456 |

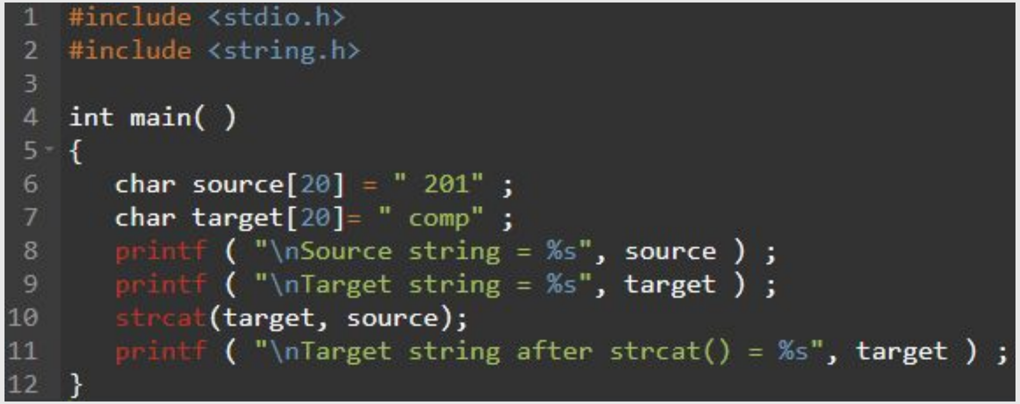
## Standard string functions in C

KOÇ UNIVERSITY

# strcat( )

- Concatenates two given strings.
- Concatenates source string at the end of destination string.

- strcat ( char * destination, char * source );

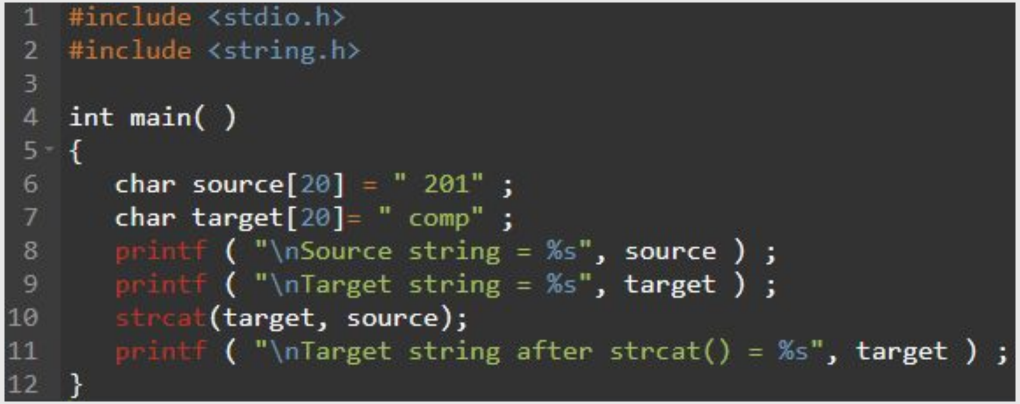


```
#include <stdio.h>
#include <string.h>

int main( )
{
   char source[20] = " 201" ;
   char target[20]= " comp" ;
   printf ( "\nSource string = %s", source ) ;
   printf ( "\nTarget string = %s", target ) ;
   strcat(target, source);
   printf ( "\nTarget string after strcat() = %s", target ) ;
}
```

## strcat( )

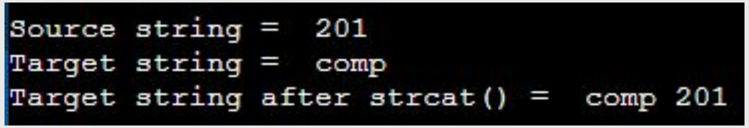
- Concatenates two given strings.
- Concatenates source string at the end of destination string.
- strcat ( char * destination, char * source );



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

int main( )
{
    char source[20] = " 201" ;
    char target[20]= " comp" ;
    printf ( "\nSource string = %s", source ) ;
    printf ( "\nTarget string = %s", target ) ;
    strcat(target, source);
    printf ( "\nTarget string after strcat() = %s", target ) ;
}
```

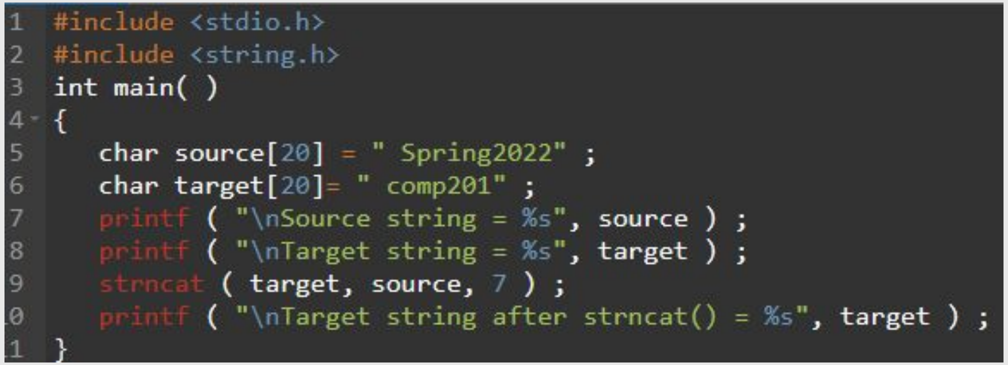
Output:



```
Source string =  201
Target string =  comp
Target string after strcat() =  comp 201
```

## strncat( )

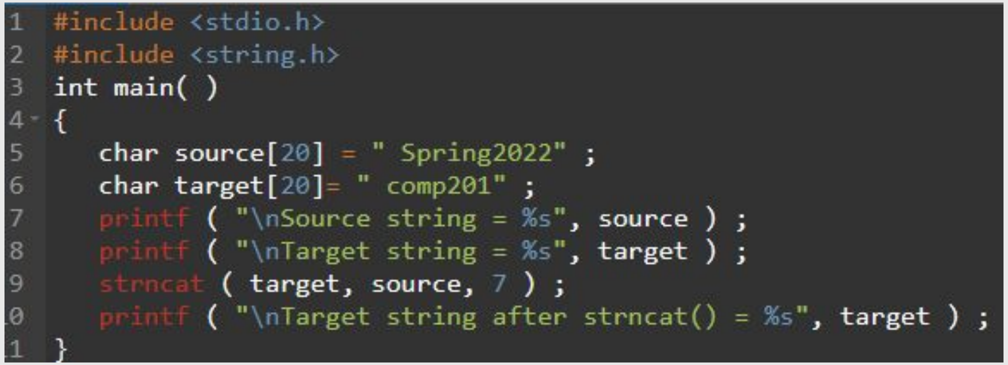
- Concatenates (appends) portion of one string at the end of another string.
- strncat ( char * destination, char * source, size_t num );



```
#include <stdio.h>
#include <string.h>
int main( )
{
    char source[20] = " Spring2022" ;
    char target[20]= " comp201" ;
    printf ( "\nSource string = %s", source ) ;
    printf ( "\nTarget string = %s", target ) ;
    strncat ( target, source, 7 ) ;
    printf ( "\nTarget string after strncat() = %s", target ) ;
}
```

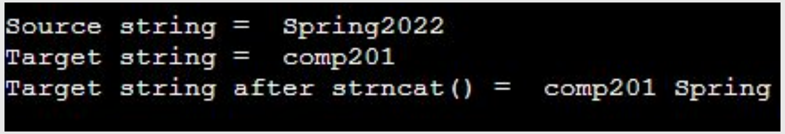
## strncat( )

- Concatenates (appends) portion of one string at the end of another string.
- strncat ( char * destination, char * source, size_t num );



```c
#include <stdio.h>
#include <string.h>
int main( )
{
    char source[20] = " Spring2022" ;
    char target[20]= " comp201" ;
    printf ( "\nSource string = %s", source ) ;
    printf ( "\nTarget string = %s", target ) ;
    strncat ( target, source, 7 ) ;
    printf ( "\nTarget string after strncat() = %s", target ) ;
}
```

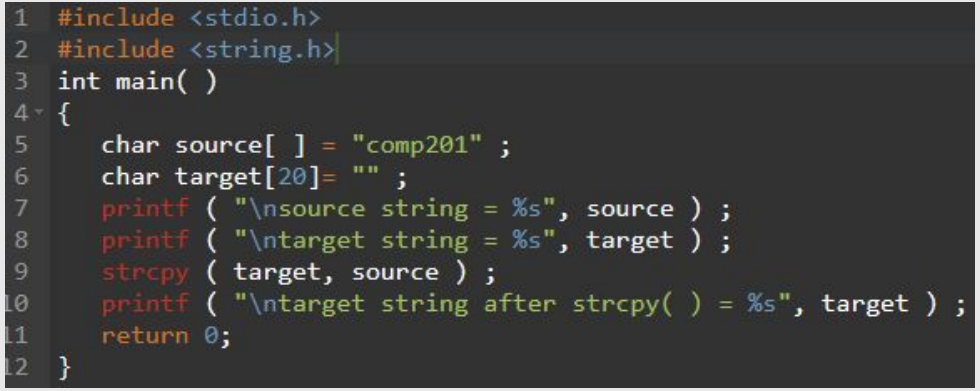
Output:



```
Source string =  Spring2022
Target string =  comp201
Target string after strncat() =  comp201 Spring
```

## strcpy( )

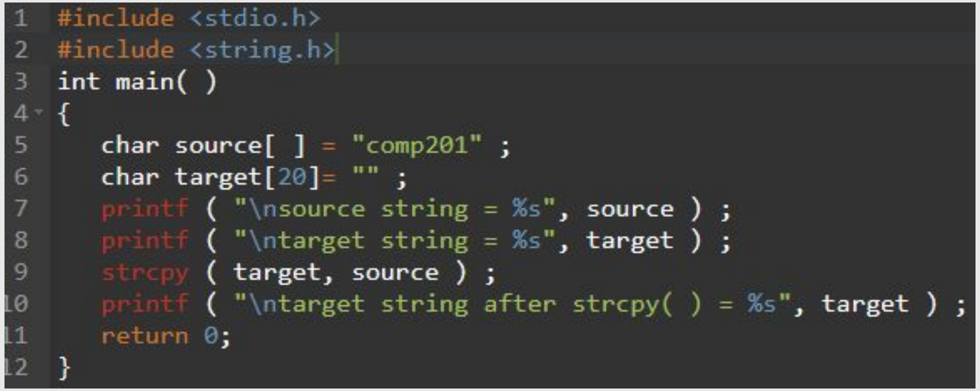
- Copies contents of one string into another string.
- strcpy ( char * destination, char * source );



```c
#include <stdio.h>
#include <string.h>
int main( )
{
    char source[ ] = "comp201" ;
    char target[20]= "" ;
    printf ( "\nsource string = %s", source ) ;
    printf ( "\ntarget string = %s", target ) ;
    strcpy ( target, source ) ;
    printf ( "\ntarget string after strcpy( ) = %s", target ) ;
    return 0;
}
```

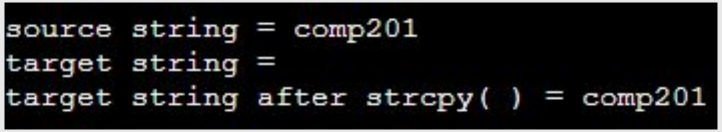
# strcpy( )

- Copies contents of one string into another string.
- strcpy ( char * destination, char * source );



```c
#include <stdio.h>
#include <string.h>
int main( )
{
    char source[ ] = "comp201" ;
    char target[20]= "" ;
    printf ( "\nsource string = %s", source ) ;
    printf ( "\ntarget string = %s", target ) ;
    strcpy ( target, source ) ;
    printf ( "\ntarget string after strcpy( ) = %s", target ) ;
    return 0;
}
```

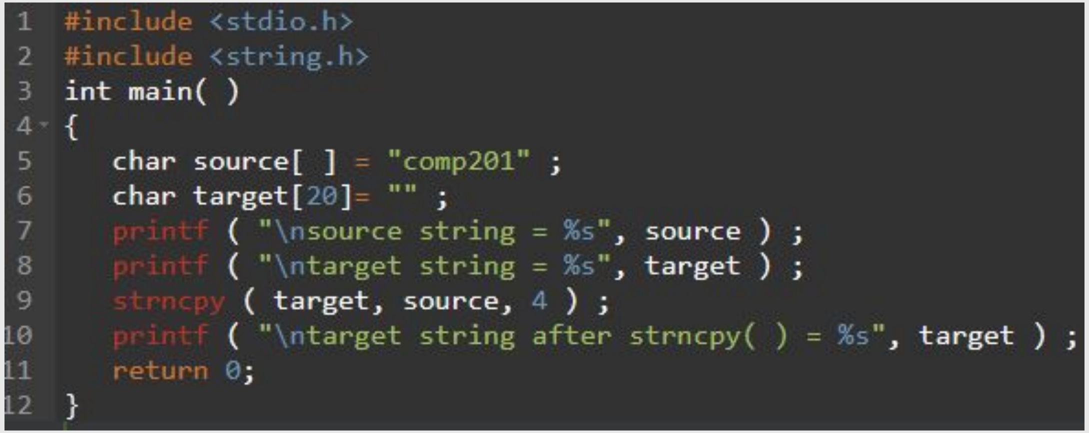
Output:



```
source string = comp201
target string =
target string after strcpy( ) = comp201
```

## strncpy( )

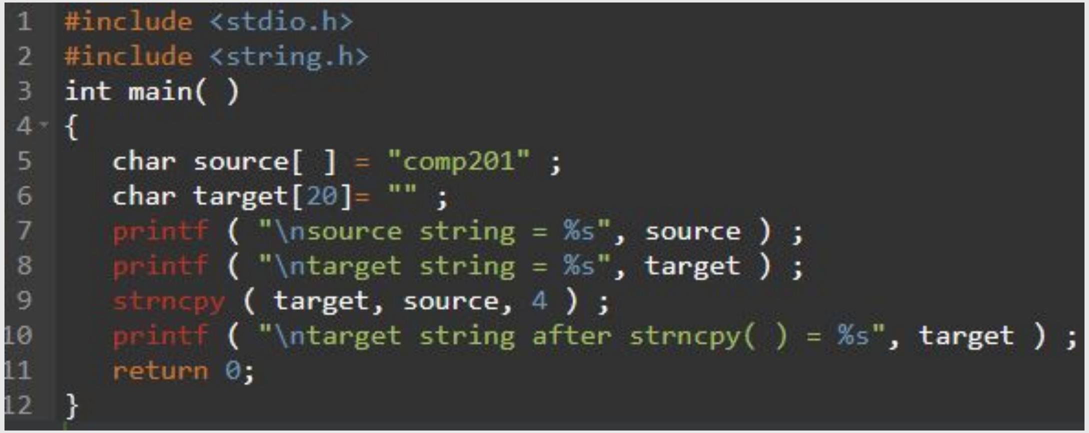
- Copies portion of contents of one string into another string.
- strncpy ( char * destination, char * source, size_t num );



```
#include <stdio.h>
#include <string.h>
int main( )
{
    char source[ ] = "comp201" ;
    char target[20]= "" ;
    printf ( "\nsource string = %s", source ) ;
    printf ( "\ntarget string = %s", target ) ;
    strncpy ( target, source, 4 ) ;
    printf ( "\ntarget string after strncpy( ) = %s", target ) ;
    return 0;
}
```

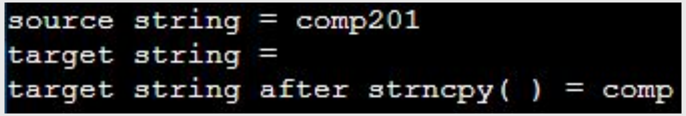
# strncpy( )

- Copies portion of contents of one string into another string.
- strncpy ( char * destination, char * source, size_t num );



```c
#include <stdio.h>
#include <string.h>
int main( )
{
    char source[ ] = "comp201" ;
    char target[20]= "" ;
    printf ( "\nsource string = %s", source ) ;
    printf ( "\ntarget string = %s", target ) ;
    strncpy ( target, source, 4 ) ;
    printf ( "\ntarget string after strncpy( ) = %s", target ) ;
    return 0;
}
```

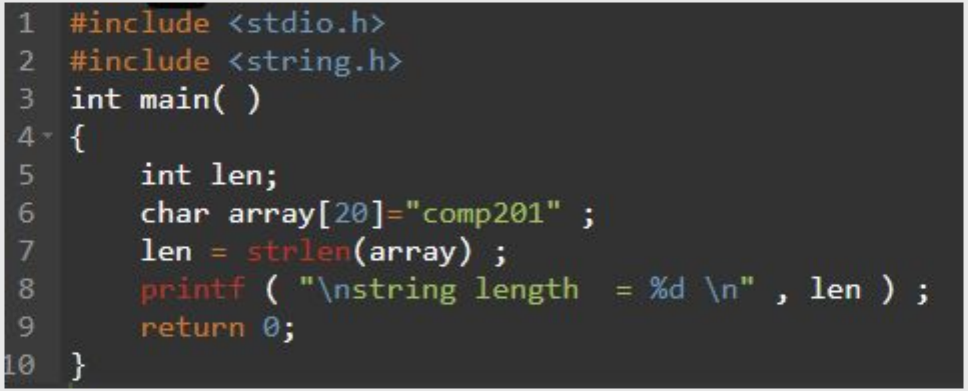
Output:



```
source string = comp201
target string =
target string after strncpy( ) = comp
```

## strlen( )

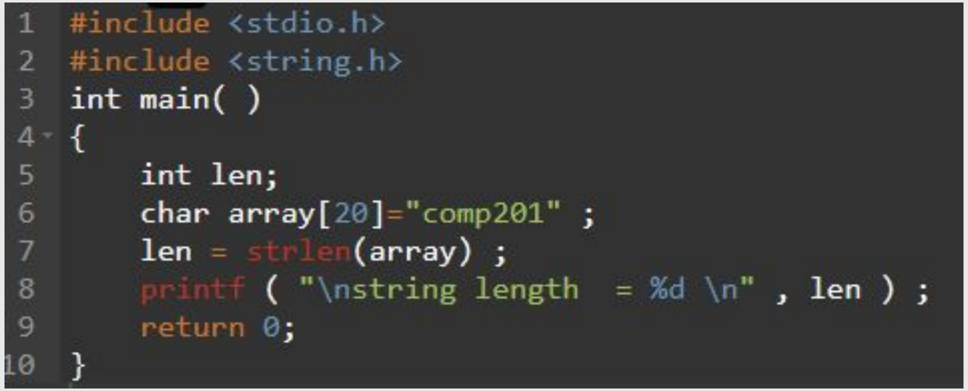
- Gives the length of the given string.
- strlen (char * str );


```c
#include <stdio.h>
#include <string.h>
int main( )
{
    int len;
    char array[20]="comp201" ;
    len = strlen(array) ;
    printf ( "\nstring length  = %d \n" , len ) ;
    return 0;
}
```

KOÇ UNIVERSITY

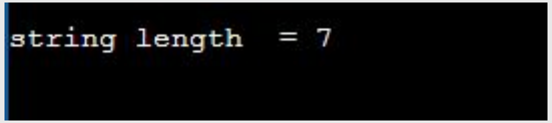
## strlen( )

- Gives the length of the given string.
- strlen (char * str );



```c
#include <stdio.h>
#include <string.h>
int main( )
{
    int len;
    char array[20]="comp201" ;
    len = strlen(array) ;
    printf ( "\nstring length  = %d \n" , len ) ;
    return 0;
}
```

Output:

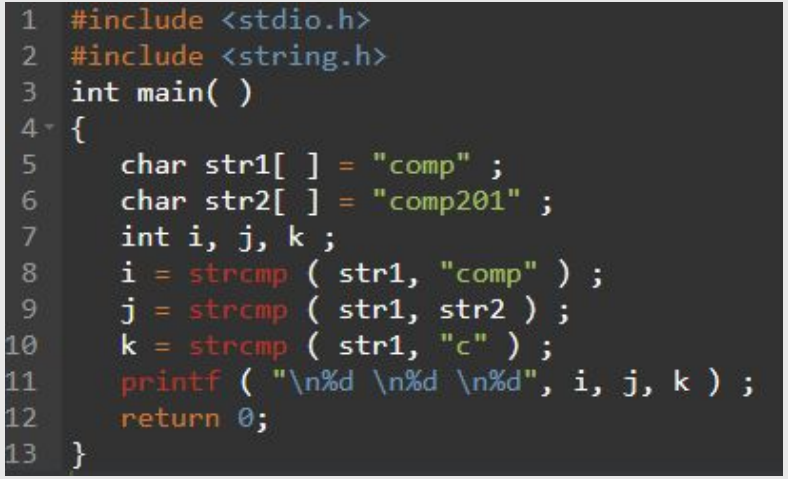


```
string length  = 7
```

KOÇ UNIVERSITY

# strcmp( )

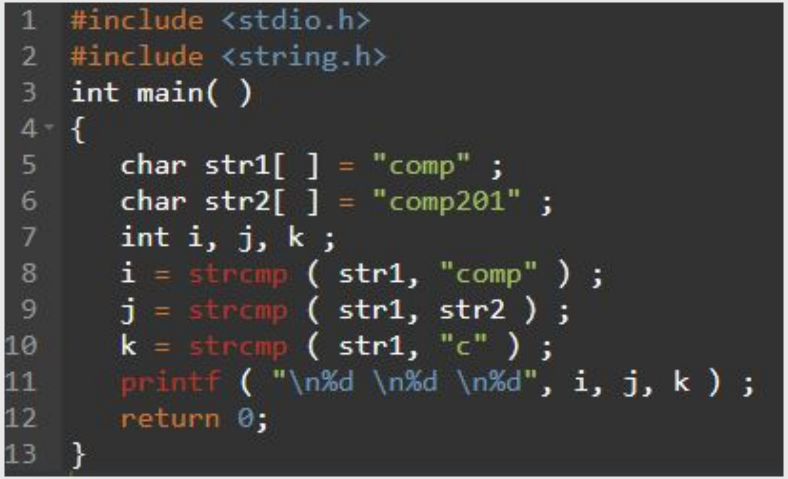
- Compares two given strings and returns zero if they are same.
- If length of string1 < string2, it returns < 0 value.
- If length of string1 > string2, it returns > 0 value.
- strcmp (char * str1, char * str2 );



```c
#include <stdio.h>
#include <string.h>
int main( )
{
    char str1[ ] = "comp" ;
    char str2[ ] = "comp201" ;
    int i, j, k ;
    i = strcmp ( str1, "comp" ) ;
    j = strcmp ( str1, str2 ) ;
    k = strcmp ( str1, "c" ) ;
    printf ( "\n%d \n%d \n%d", i, j, k ) ;
    return 0;
}
```

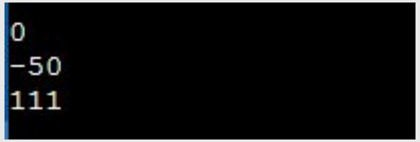
## strcmp( )

- Compares two given strings and returns zero if they are same.
- If length of string1 < string2, it returns < 0 value.
- If length of string1 > string2, it returns > 0 value.
- strcmp (char * str1, char * str2 );



```c
#include <stdio.h>
#include <string.h>
int main( )
{
    char str1[ ] = "comp" ;
    char str2[ ] = "comp201" ;
    int i, j, k ;
    i = strcmp ( str1, "comp" ) ;
    j = strcmp ( str1, str2 ) ;
    k = strcmp ( str1, "c" ) ;
    printf ( "\n%d \n%d \n%d", i, j, k ) ;
    return 0;
}
```

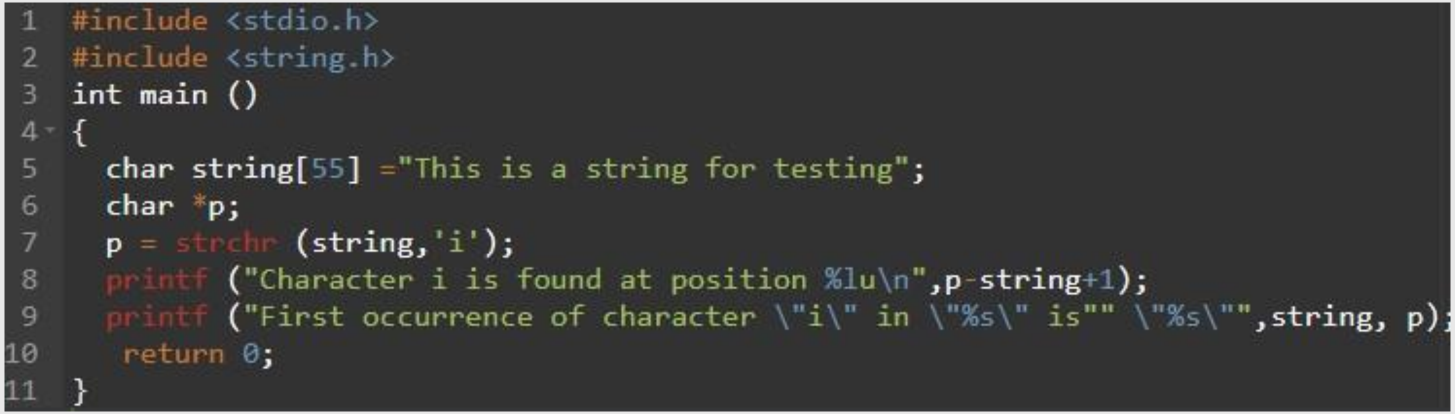
Output:



```
0
-50
111
```

## strchr( )

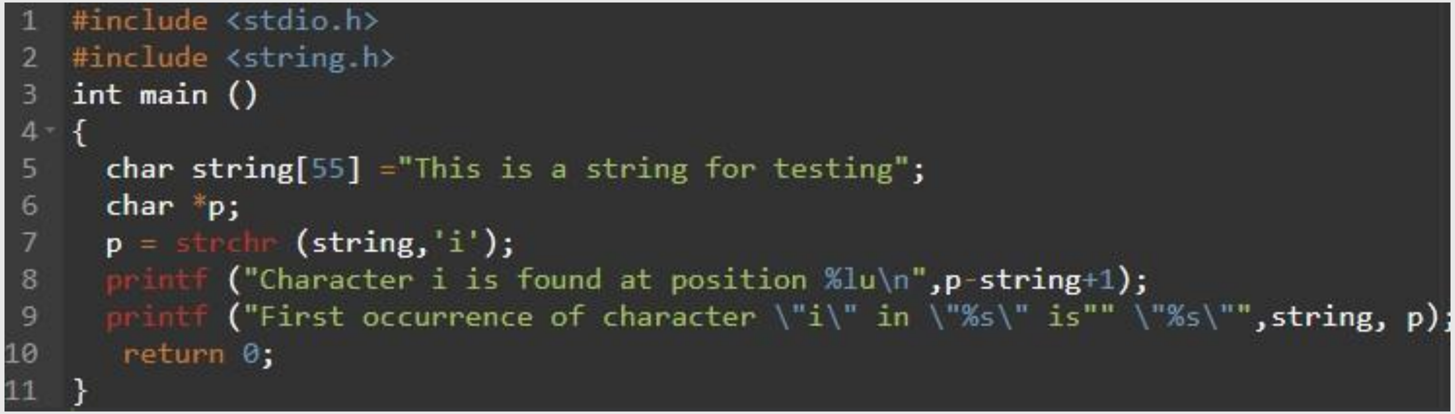
- Returns pointer to the first occurrence of the character in a given string.
- strchr(char *str, character);



```c
#include <stdio.h>
#include <string.h>
int main ()
{
  char string[55] ="This is a string for testing";
  char *p;
  p = strchr (string,'i');
  printf ("Character i is found at position %lu\n",p-string+1);
  printf ("First occurrence of character \"i\" in \"%s\" is"" \"%s\"",string, p);
   return 0;
}
```

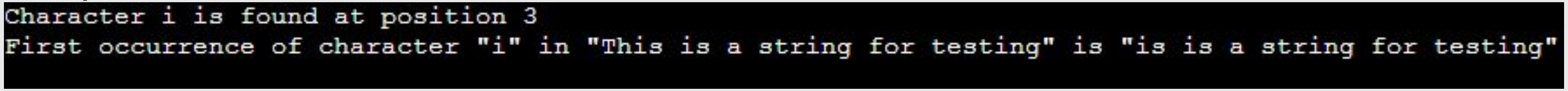
## strchr( )

- Returns pointer to the first occurrence of the character in a given string.
- strchr(char *str, character);



```c
#include <stdio.h>
#include <string.h>
int main ()
{
  char string[55] ="This is a string for testing";
  char *p;
  p = strchr (string,'i');
  printf ("Character i is found at position %lu\n",p-string+1);
  printf ("First occurrence of character \"i\" in \"%s\" is"" \"%s\"",string, p);
   return 0;
}
```

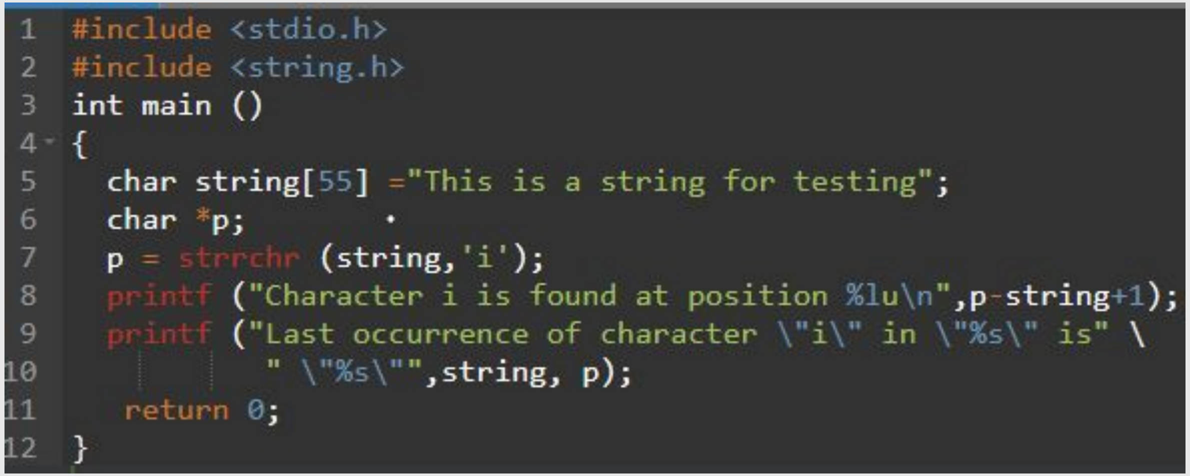
Output:



```
Character i is found at position 3
First occurrence of character "i" in "This is a string for testing" is "is is a string for testing"
```

## strrchr( )

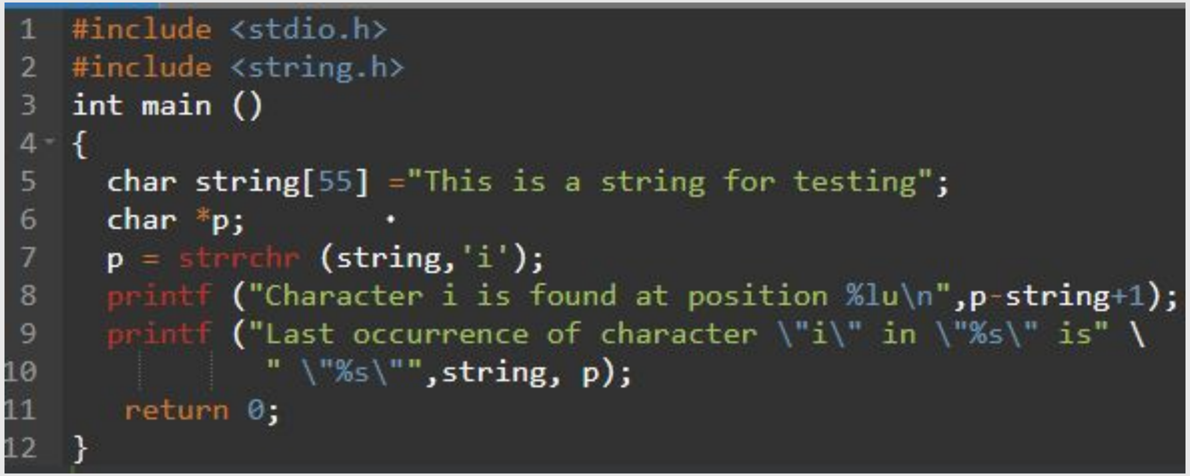
- Returns pointer to the last occurrence of the character in a given string.
- strrchr(char *str, character);



```c
#include <stdio.h>
#include <string.h>
int main ()
{
   char string[55] ="This is a string for testing";
   char *p;
   p = strrchr (string,'i');
   printf ("Character i is found at position %lu\n",p-string+1);
   printf ("Last occurrence of character \"i\" in \"%s\" is" \
           " \"%s\"",string, p);
    return 0;
}
```

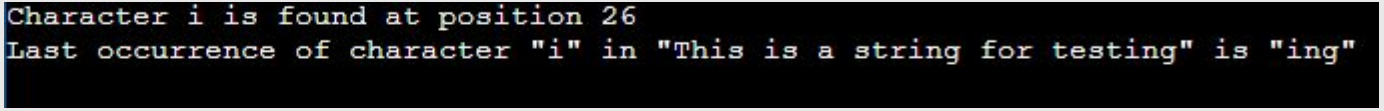
## strrchr( )

- Returns pointer to the last occurrence of the character in a given string.
- strrchr(char *str, character);



```c
#include <stdio.h>
#include <string.h>
int main ()
{
  char string[55] ="This is a string for testing";
  char *p;
  p = strrchr (string,'i');
  printf ("Character i is found at position %lu\n",p-string+1);
  printf ("Last occurrence of character \"i\" in \"%s\" is" \
        " \"%s\"",string, p);
   return 0;
}
```

Output:

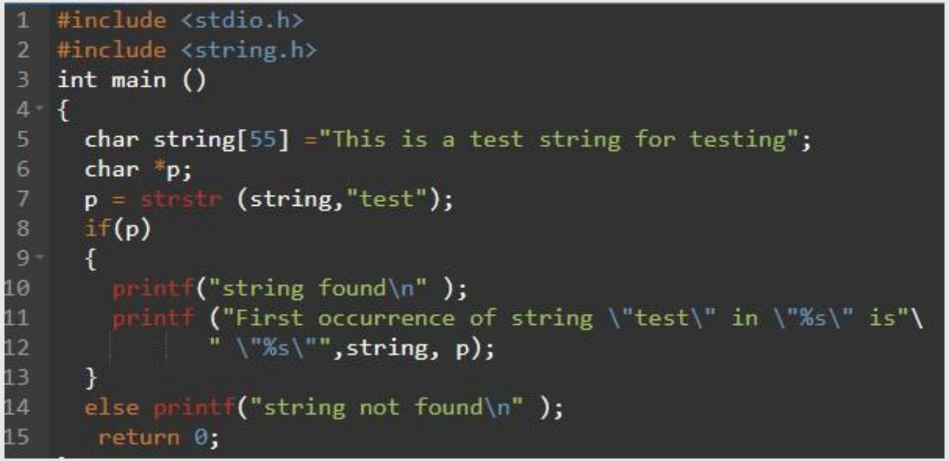


```
Character i is found at position 26
Last occurrence of character "i" in "This is a string for testing" is "ing"
```

KOÇ UNIVERSITY

## strstr( )

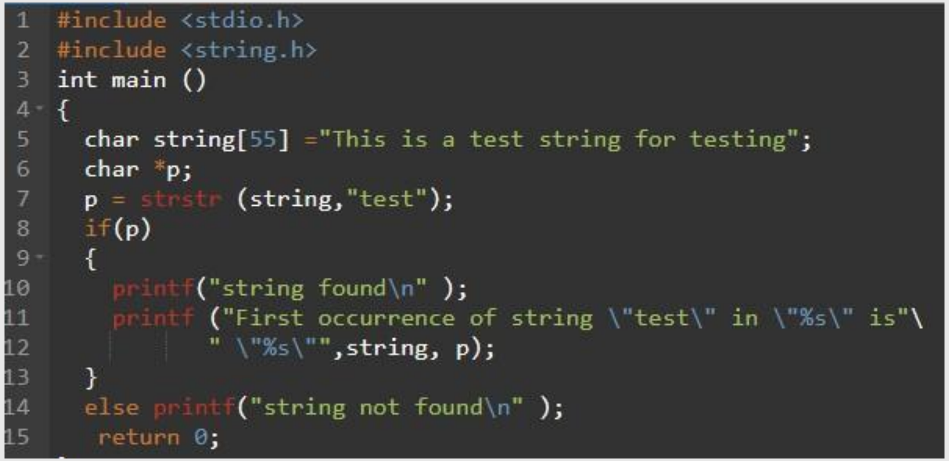
- Returns pointer to the first occurrence of the string in a given string.
- strstr(char *str1, char *str2);



```c
#include <stdio.h>
#include <string.h>
int main ()
{
  char string[55] ="This is a test string for testing";
  char *p;
  p = strstr (string,"test");
  if(p)
  {
    printf("string found\n" );
    printf ("First occurrence of string \"test\" in \"%s\" is"\
           " \"%s\"",string, p);
  }
  else printf("string not found\n" );
   return 0;
```

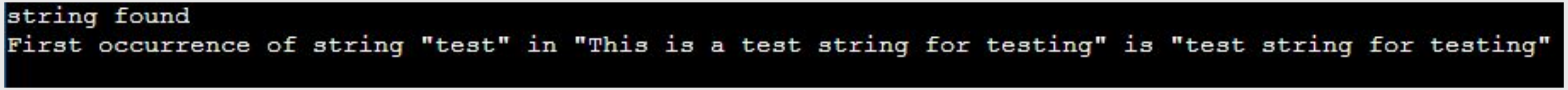
# strstr( )

- Returns pointer to the first occurrence of the string in a given string.
- strstr(char *str1, char *str2);



```c
#include <stdio.h>
#include <string.h>
int main ()
{
  char string[55] ="This is a test string for testing";
  char *p;
  p = strstr (string,"test");
  if(p)
  {
    printf("string found\n" );
    printf ("First occurrence of string \"test\" in \"%s\" is"\
            " \"%s\"",string, p);
  }
  else printf("string not found\n" );
   return 0;
```

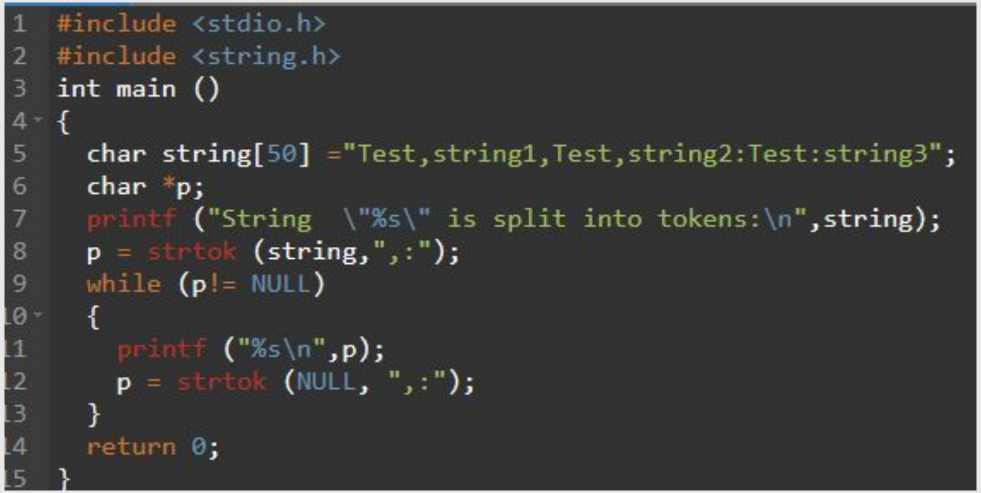
Output:



```
string found
First occurrence of string "test" in "This is a test string for testing" is "test string for testing"
```

# strtok( )

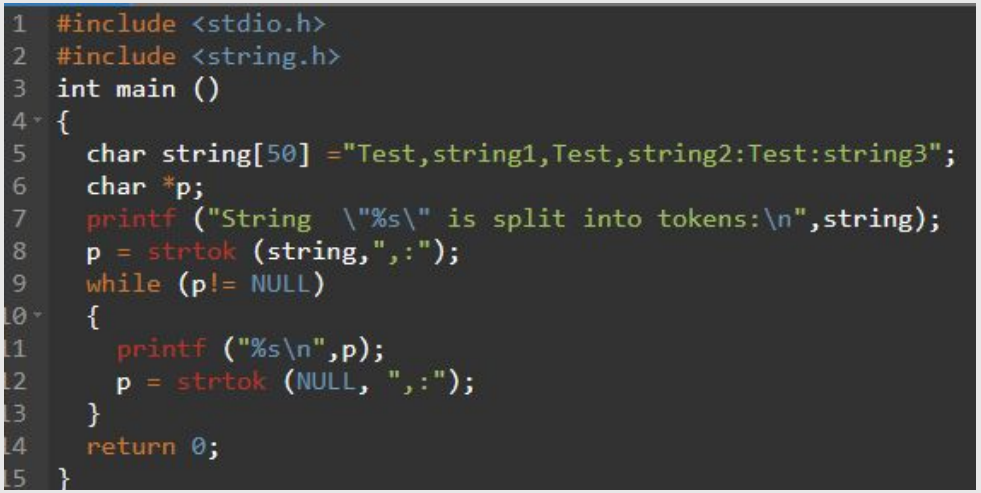
- Tokenizes/parses the given string using delimiter.
- strtok ( char * str, char * delimiters );



```c
#include <stdio.h>
#include <string.h>
int main ()
{
  char string[50] ="Test,string1,Test,string2:Test:string3";
  char *p;
  printf ("String  \"%s\" is split into tokens:\n",string);
  p = strtok (string,",:");
  while (p!= NULL)
  {
    printf ("%s\n",p);
    p = strtok (NULL, ",:");
  }
  return 0;
}
```

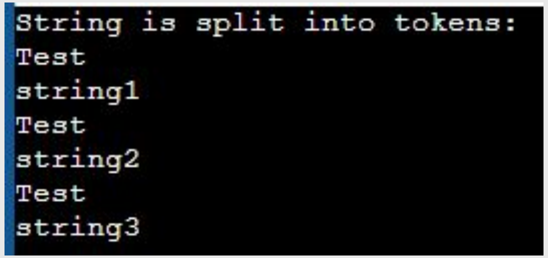
# strtok( )

- Tokenizes/parses the given string using delimiter.
- strtok ( char * str, char * delimiters );



```c
#include <stdio.h>
#include <string.h>
int main ()
{
  char string[50] ="Test,string1,Test,string2:Test:string3";
  char *p;
  printf ("String  \"%s\" is split into tokens:\n",string);
  p = strtok (string,",:");
  while (p!= NULL)
  {
    printf ("%s\n",p);
    p = strtok (NULL, ",:");
  }
  return 0;
}
```

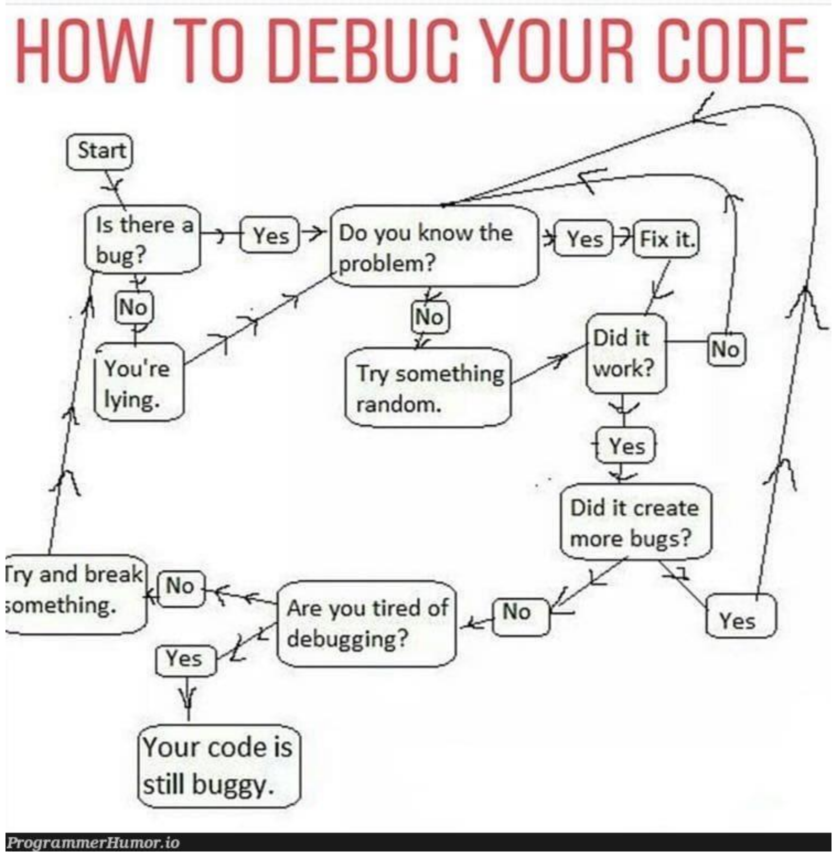
Output:



```
String is split into tokens:
Test
string1
Test
string2
Test
string3
```

# COMP201-Spring26

Lab 2: Debugging



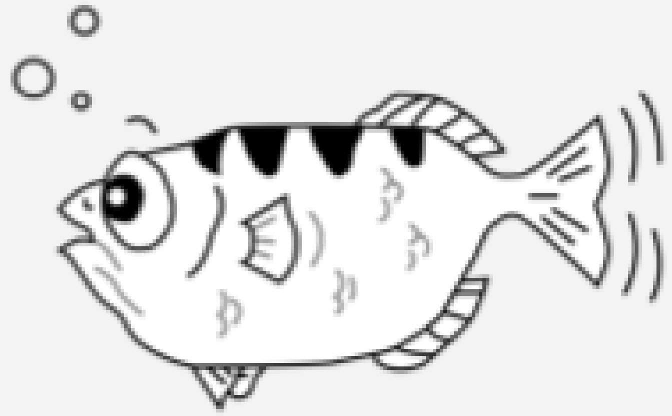
## Debugging

```
console.log("got here");
console.log("aaaaaa");
console.log("bbbbbb");
```

**Printf Debugging and Logging**

> *The most effective debugging tool is still careful thought, coupled with judiciously placed print statements*
>
> Brian Kernighan, *Unix for Beginners*.

## What is GNU Debugger (GDB)?



www.gnu.org/software/gdb

“GNU Debugger” is a debugger for several languages, including C and C++. It was first written by Richard Stallman in 1986 as part of his GNU system.

- It allows you to inspect what the program is doing at a certain point during execution.
- Errors like segmentation faults may be easier to find with the help of gdb.

KOÇ UNIVERSITY

## Start with GDB

When compiling you need to add a “-g” option to enable built-in debugging support :

- $ gcc [other flags] -g <source files> -o <output file>
    - e.g. : $ gcc -g main.c -o main.out

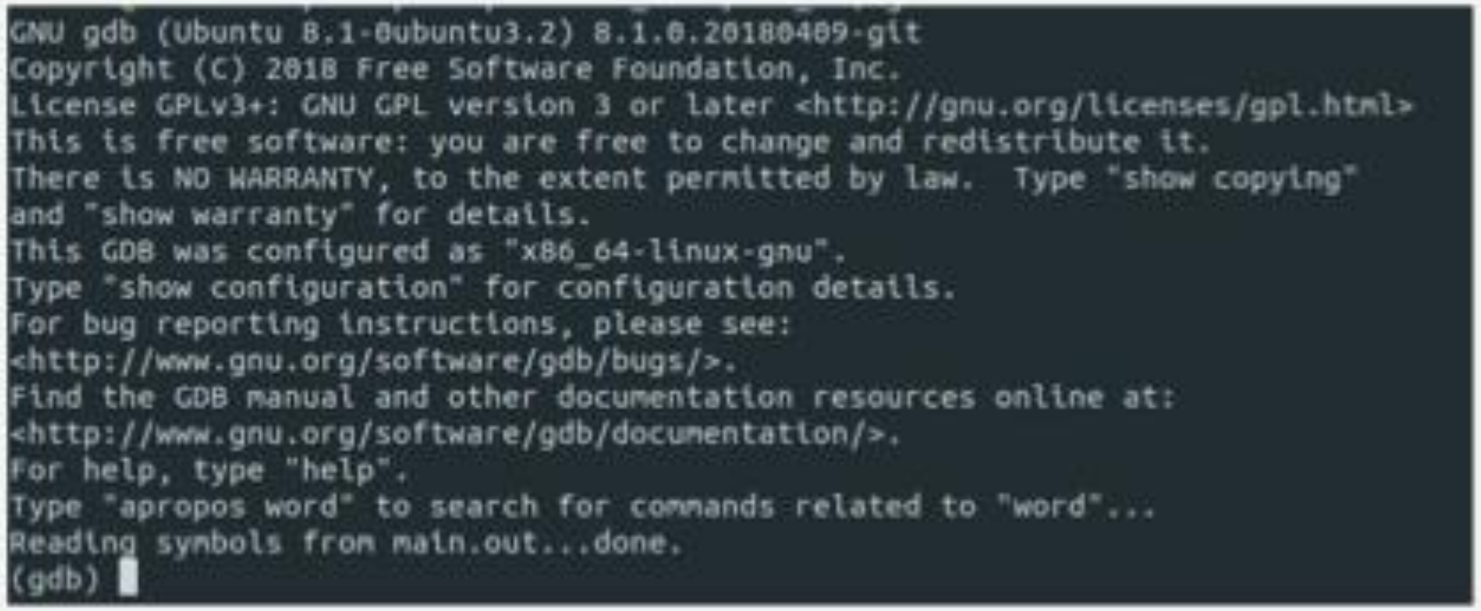
To run GDB simply run:

- $ gdb [compiled_file]
    - e.g. : $ gdb main.out

KOÇ UNIVERSITY

## GDB prompt

- `GDB` has an interactive shell, much like the linux terminals. It can recall history with the arrow keys, auto-complete words (most of the time) with the TAB key, and has other nice features like help command ( help [command] ) that provides more details of the command.



```
GNU gdb (Ubuntu 8.1-0ubuntu3.2) 8.1.0.20180409-git
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
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"...
Reading symbols from main.out...done.
(gdb) 
```

KOÇ UNIVERSITY

## Running a program in GDB

To run the program, just use

- (gdb) run

This runs the program

- If it has no serious problems (i.e. the normal program didn't get a segmentation fault, etc.), the program should run fine here too.

- If the program did have issues, then you (should) get some useful information like the line number where it crashed, and parameters to the function that caused the error:

KOÇ UNIVERSITY

## Line Breakpoint

Running the program in case it's buggy or crashes may not be that useful, so instead you can set a breakpoint, a line of the code your debugger stops there and you can run the code line by line while you can observe variables.

To set a breakpoint use ( let's say your source code file is named my_file.c)

- (gdb) break my_file.c:7

  or
- (gdb) b  my_file.c:7

This sets a breakpoint in line 7 of the my_file.c code and when you run the code, if the program ever reaches line 7, the debugger stops there.

## Function Name Breakpoint

You can also tell gdb to break at a particular function. Suppose you have a function named "myfunc" then you can set a breakpoint for whenever this function is called using

- (gdb) break myfunc
  or
- (gdb) break myfunc

KOÇ UNIVERSITY

## What’s next?

After setting suitable Breakpoints and running the program using “ (gdb) run “ command, it should stop where you tell it to. You can proceed onto the next breakpoint by typing

- (gdb) continue

Or you can execute a single-step (execute just the next line of code) by typing

- (gdb) step — step into

  or
- (gdb) next

  or
- (gdb) n

  $\}$ step over

  or
- Simply press ENTER after once you called next, the ENTER executes the last command again

KOÇ UNIVERSITY

## How to check variables?

If you reached to a desired point and you want to see things like the values of variables, etc.

The “print” command prints the value of the specified variable

- (gdb) print [variable]
- e.g. : (gdb) print my_var

And “print/x” prints the value in hexadecimal and “print/t” in binary format

- (gdb) print/x [variable]
- (gdb) print/t [variable]

KOÇ UNIVERSITY

## How to watch for any change?

Whereas breakpoints interrupt the program at a particular line or function, watchpoints act on variables. They pause the program whenever a watched variable's value is modified. For example, the following watch command:

- (gdb) watch [variable]
- e.g.  : (gdb) watch my_var

Whenever my_var's value is modified, the program will interrupt and print out the old and new values.

## Conditional breakpoint!

Just like regular breakpoints, except that you get to specify some criterion that must be met for the breakpoint to trigger.

- (gdb) break my_file.c:6 if i >= ARRAYSIZE

This command sets a breakpoint at line 6 of file my_file.c, which triggers only if the variable “i” is greater than or equal to the size of the array (suppose ARRAYSIZE is defined). Conditional breakpoints can most likely avoid all the unnecessary steppings and time wastings.

KOÇ UNIVERSITY

## More useful commands

Other useful commands:

- **(gdb) backtrace** produces a stack trace of the function calls that lead to a seg fault (similar to Java exceptions)
- **(gdb) where** same as backtrace; you can think of this version as working even when you're still in the middle of the program
- **(gdb) finish** runs until the current function is finished
- **(gdb) delete** deletes a specified breakpoint
- **(gdb) info breakpoints** shows information about all declared breakpoints
- **(gdb) info locals** shows local variables and their values in the current score

KOÇ UNIVERSITY

## AI for Debugging

Where LLMs shine:

- Explaining cryptic error messages
- Traversing language and abstraction boundaries
- Correlating symptoms with root causes
- Analyzing crash dumps and stack traces

# AI for Debugging

Limitations to keep in mind:

- LLMs can hallucinate **plausible-sounding but wrong** explanations
- They may suggest fixes that **mask** the bug rather than fix it
- Always verify suggestions with actual debugging tools
- They work best as a **complement** to, not replacement for, understanding your code