

Advanced C Programming

ALAIN HOUELLE

V3.0

Compilation

C memory mapping

Preprocessor

Logic operators

Variable type & Standard data format

- ASCII

- UTF8

- INT(unsigned, signed)

- Floating point

Array, Multidimensional array, Structure, Union, enum

Pointers

Pointers on functions

Input/Output (File)

Qsort Algorithm

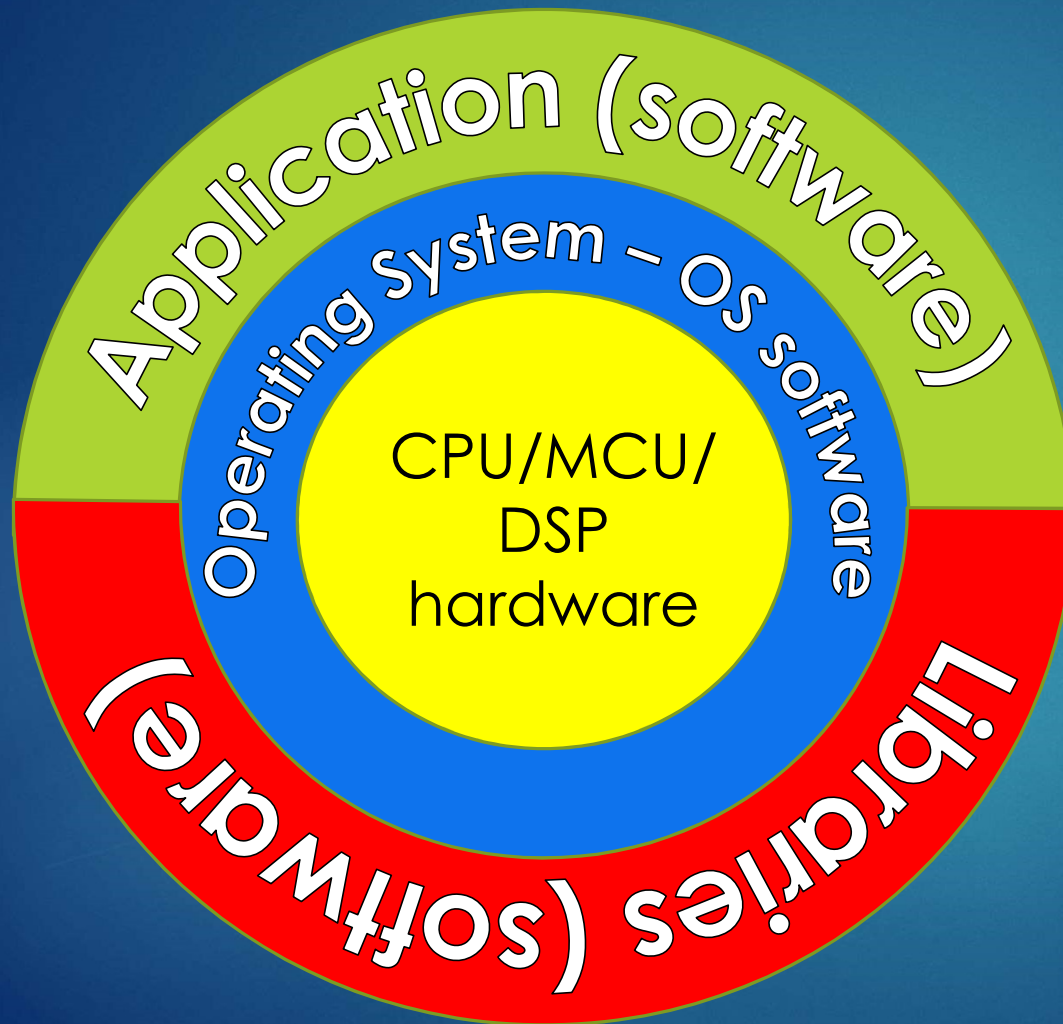
Process

Threads

Sockets (client/server)

Game competition (for the most advanced coders)

C to Exe



Most used software programming languages:

- C
 - C++
 - Java
 - Assembler
-
- A lot of Micro Controllers applications don't use the OS layer.

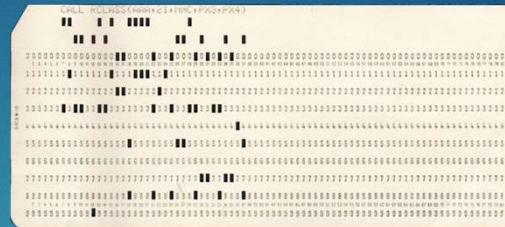
ACP

Assembler

5

Assembler

Human writable/readable
Machine language



Hexadecimal
Machine language

```
clrf  PIE1  
movlw d'255'  
movwf PR2
```

Assembler

```
0x1834A733  
0x907D6C88  
0x4533AFC5
```

.hex

.asm



ACP

Machine language

6

لا يحق لك
المروور

Вы не
павінны
праходзіць



Atmel

0x01010203
0x05080C15
0x22375990

你不可以
過去



MicroChip (Pic)

Each μ C speaks
its own machine
language
=> No portability



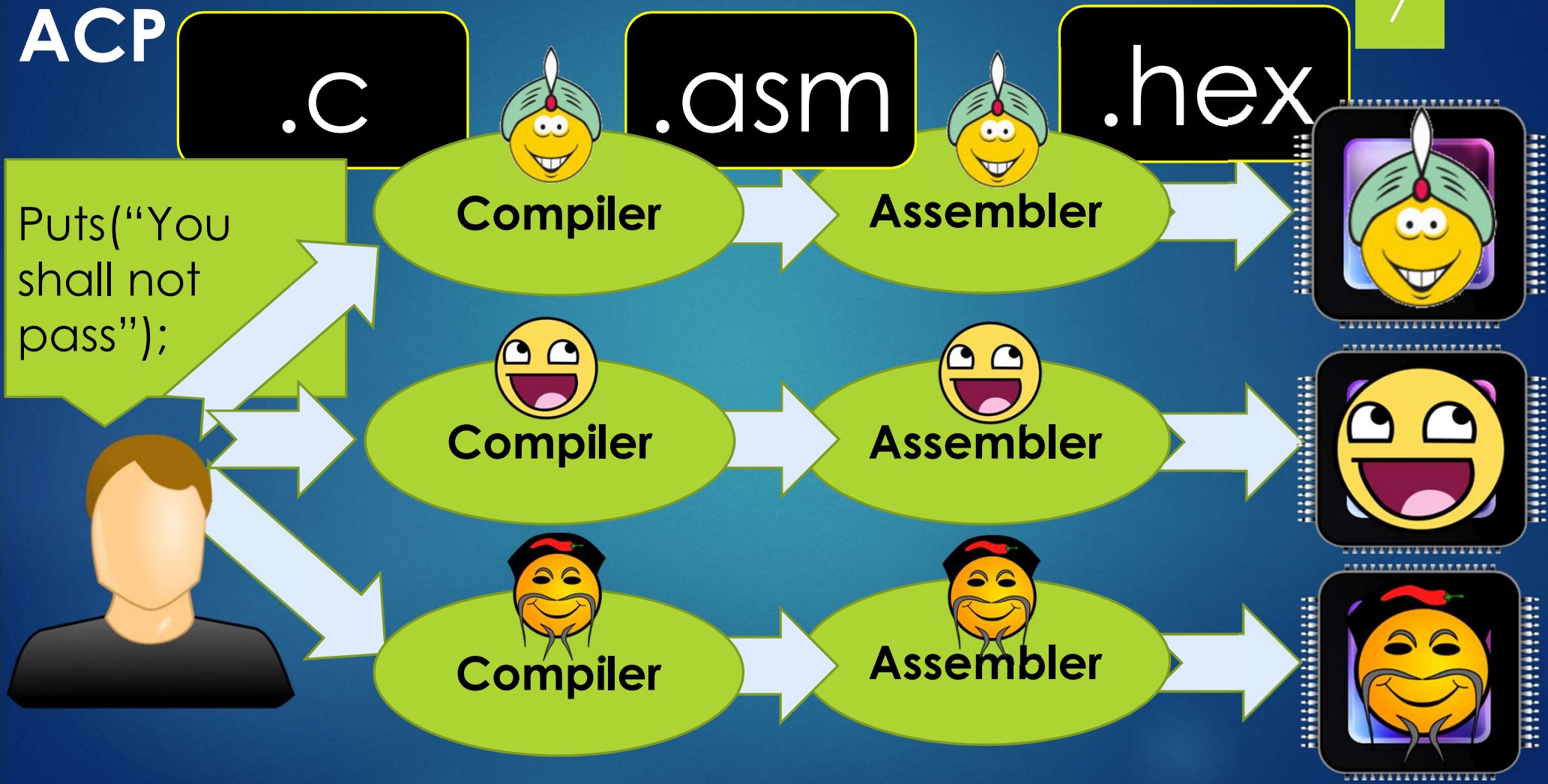
Hexadecimal



ST Micro (STM32)

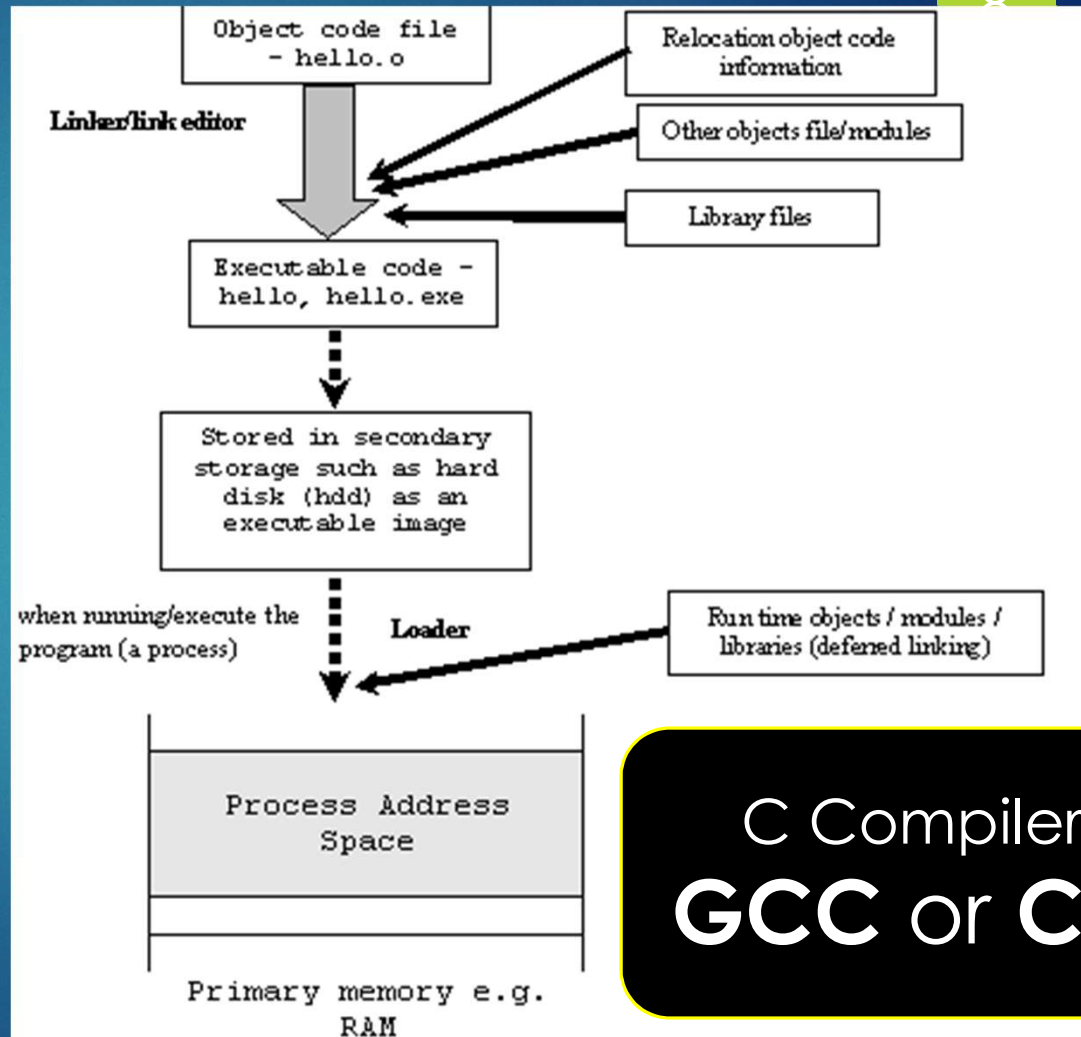
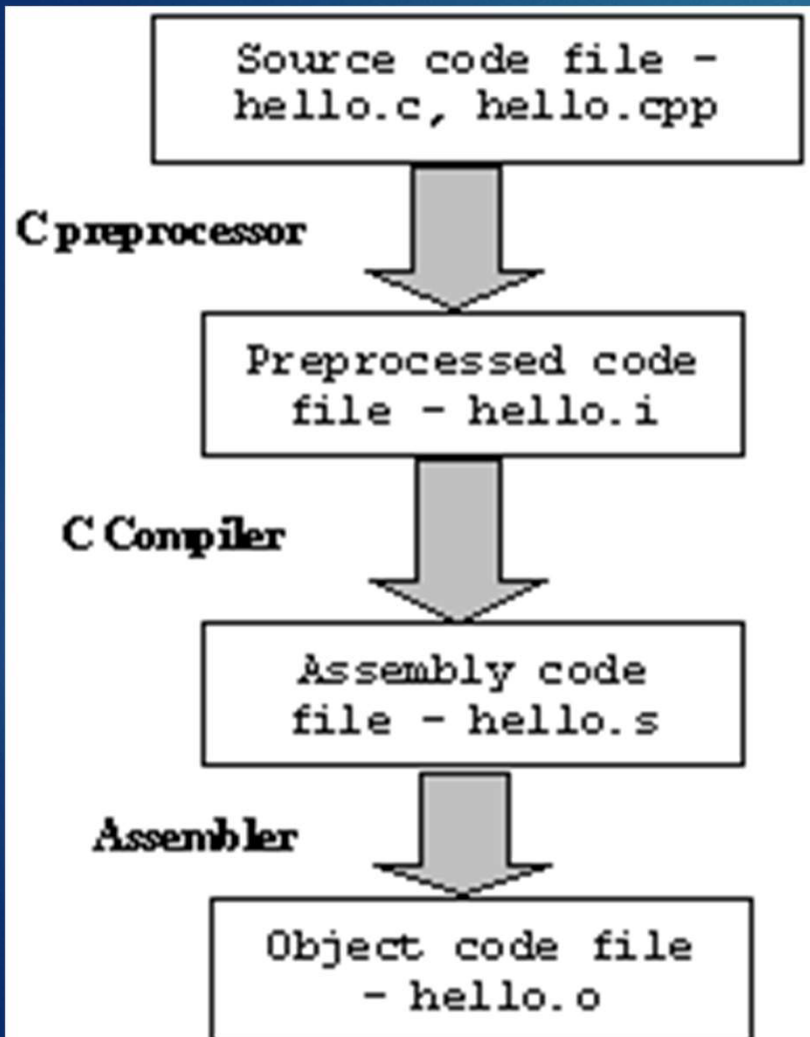
High level Programming languages

7



ACP

Compilation Stages



C Compiler
GCC or CC

ACP

test.c

Stage 1: C Preprocessor

9

```
#include "testinclude.c"
```

```
#define BB 32
```

```
int main() {  
    int question, v=12;  
    question=BB | !BB;  
    question|=v;  
#ifdef WHITERABBIT  
    puts("white rabbit is defined");  
#endif  
}
```

testinclude.c

```
int IAmNotANumberIAmAFreeMan() {  
    return 6;  
}
```

Preprocessor only:
gcc -E test.c



```
# 1 "test.c"  
# 1 "<built-in>"  
# 1 "<command-line>"  
# 1 "/usr/include/stdc-predef.h" 1 3 4  
# 1 "<command-line>" 2  
# 1 "test.c"  
# 1 "testinclude.c" 1  
int IAmNotANumberIAmAFreeMan() {  
    return 6;  
}  
# 2 "test.c" 2  
  
int main() {  
    int question, v=12;  
    question=32 | !32;  
    question|=v;  
}
```

ACP Stage 2 : Compiler

```
# 1 "test.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "/usr/include/stdc-predef.h" 1 3 4
# 1 "<command-line>" 2
# 1 "test.c"
# 1 "testinclude.c" 1
int IAmNotANumberIAmAFreeMan () {
    return 6;
}
# 2 "test.c" 2

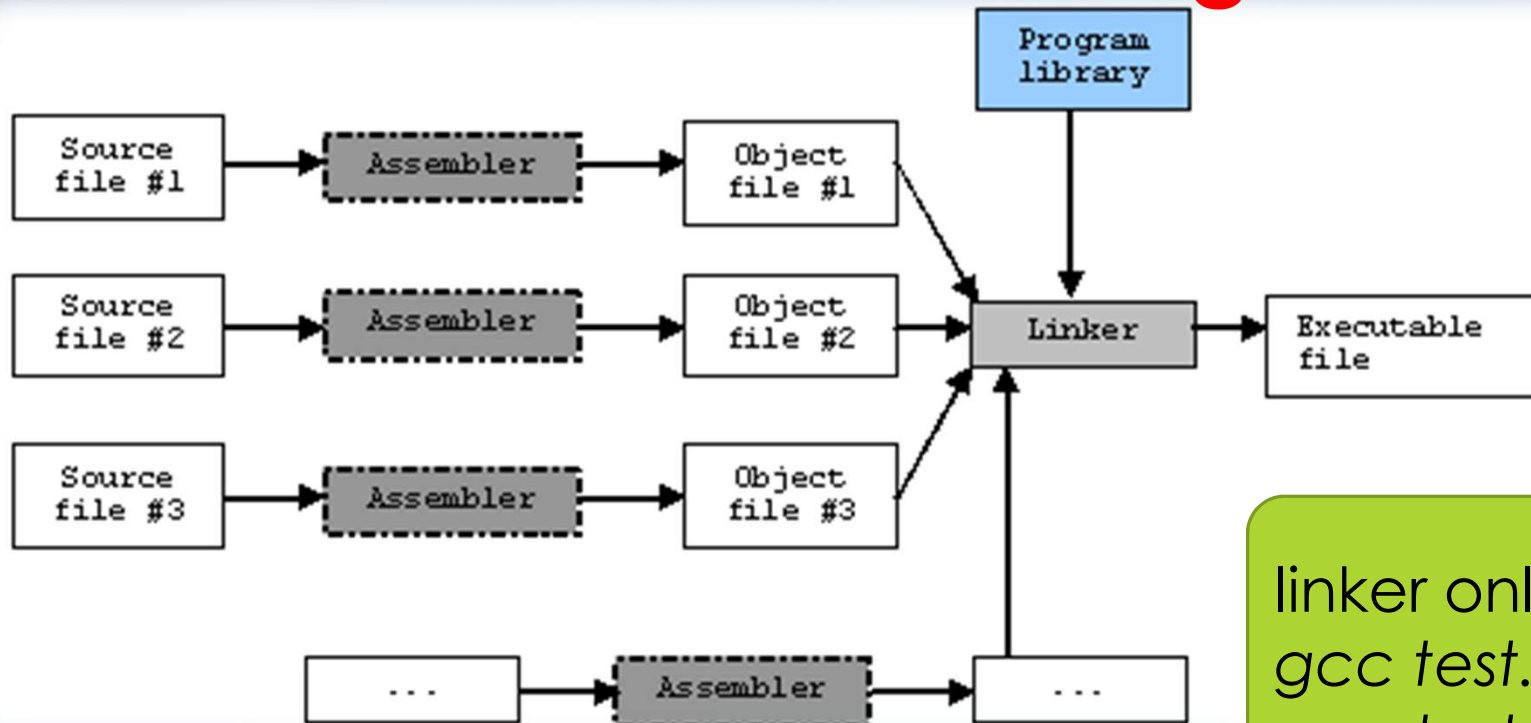
int main() {
    int question, v=12;
    question=32 | !32;
    question|=v;
}
```

Compilation only:

gcc -S test.c

gcc -S test.i

```
.file "test.c"
.text
.globl IAmNotANumberIAmAFreeMan
.type IAmNotANumberIAmAFreeMan, @function
IAmNotANumberIAmAFreeMan:
.LFB0:
    .cfi_startproc
    pushl %ebp
    .cfi_def_cfa_offset 8
    .cfi_offset 5, -8
    movl %esp, %ebp
    .cfi_def_cfa_register 5
    movl $6, %eax
    popl %ebp
    .cfi_restore 5
    .cfi_def_cfa 4, 4
    ret
    .cfi_endproc
.LFE0:
.size IAmNotANumberIAmAFreeMan, .-IAmNotANumberIAmAFreeMan
.globl main
.type main, @function
main:
.LFB1:
    .cfi_startproc
    pushl %ebp
    .cfi_def_cfa_offset 8
    .cfi_offset 5, -8
    movl %esp, %ebp
    .cfi_def_cfa_register 5
    subl $16, %esp
    movl $12, -8(%ebp)
    movl $32, -4(%ebp)
    movl -8(%ebp), %eax
    orl %eax, -4(%ebp)
    leave
    .cfi_restore 5
    .cfi_def_cfa 4, 4
    ret
    .cfi_endproc
```

linker only:
`gcc test.c -o test`
`gcc test.o -o test`

Gives executable file.
.exe on windows

PC platform :
Linux/Windows/MacOS (x86)

.C



CrossCompiler
For μ Cx



.hex

Downloader
Uploader

Target platform
 μ Cx

USB

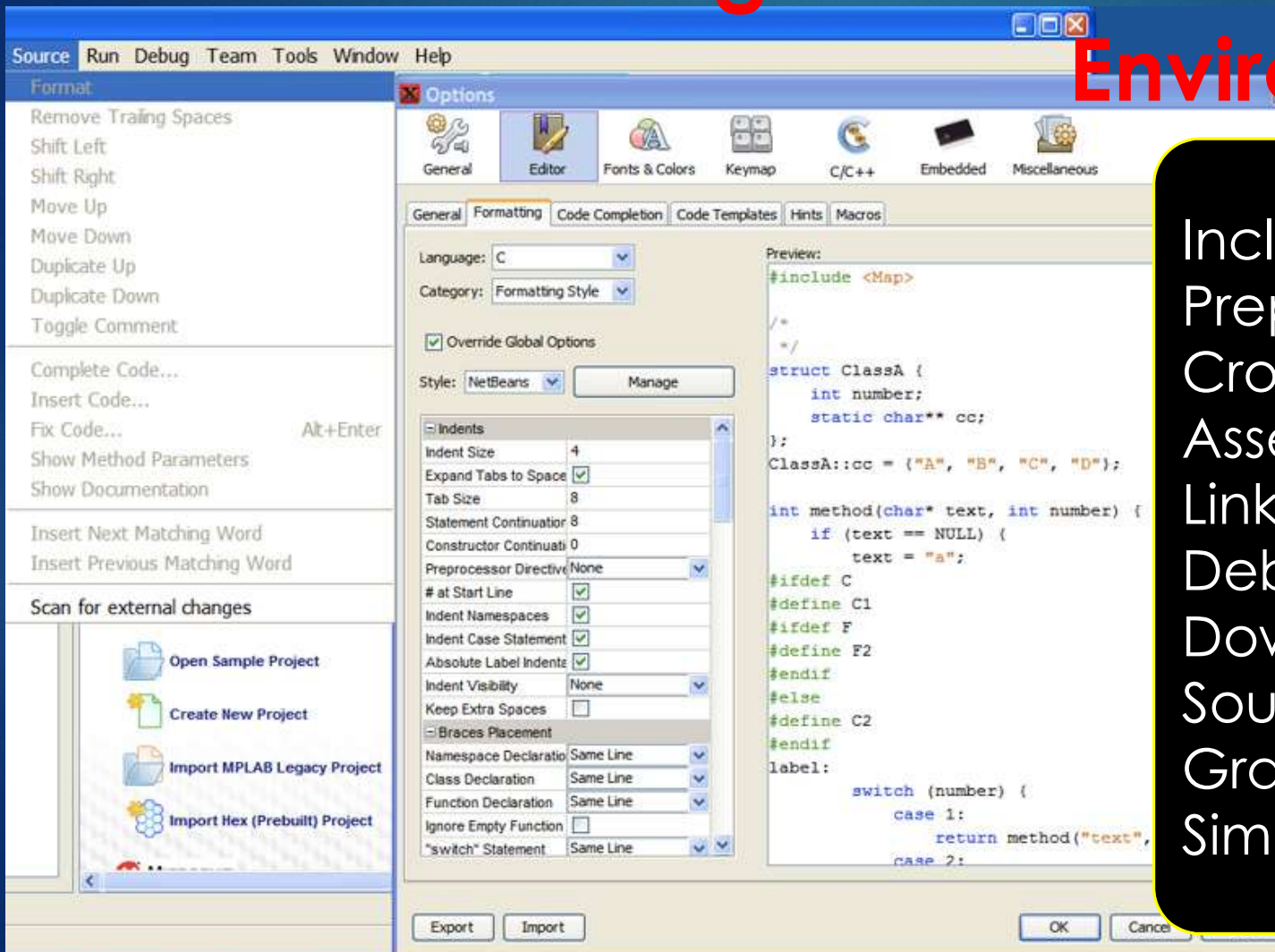
ACP

IDE : Integrated Development Environment

14

Includes:

- Preprocessor
- CrossCompiler
- Assembler
- Linker
- Debugger
- Downloader
- Source Editor
- Graphical Environment
- Simulator



C Creators

ACP

Inventors

16



Ken
Thompson

Dennis
Ritchie
1941-2011

C created in 1970 to (re)write
Unix kernel on a PDP 11
(DECsystem)

Called “C” because it was the
next version of “B” language

C

Preprocessor

```
#include <file> // include a file from the library directory
#include "file"  // include a file from the default directory (default directory is the current directory or
                // can be defined with -D gcc command line option)

#define PI 3.14159 // every variable PI in the program will be replaced by 3.14159
#define ab CD      // every ab variable will be replaced by CD

#ifdef foobar // if foobar has been defined "..." will be included in the source code, else "..." will be
...          // skipped
#endif

#define ab CD // define ab as CD
...1         // for this part of code ab will be replaced by CD
#undef ab     // undefine ab
...2         // for this part of code ab will not be replaced
```

Macro Functions:

```
#define myMacro(x) printf("Value of X=%d\n",x)
```

C code example:

```
....  
int number;  
myMacro(number);  
....
```

After preprocessing (gcc -E):

```
....  
int number;  
printf("Value of X=%d\n",number);  
....
```

C

Memory model

ACP Global Structure of a C Program

21

Preprocessor directives

#include
#define
#ifdef

Global Declarations

Variable which
can be reached
by any functions
of the program

Functions

Any function
needed by the
application

main() {...}

Main function.
The program will
start here

ACP Global Structure of a C Program

function

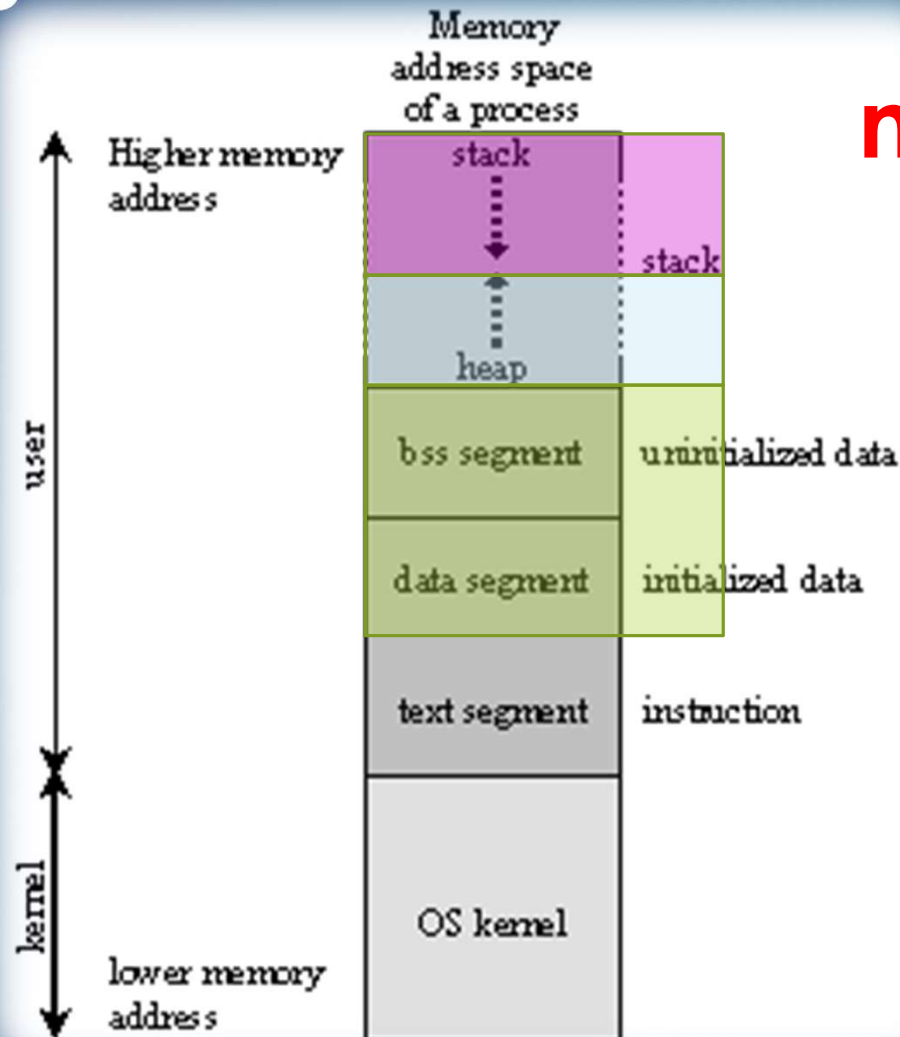
22

```
return type function(parameters) {  
    local declaration  
  
    source code  
    return value  
}
```

Local declaration only
available for the function

Local variable=dynamic
variable :
it exists only when the
function is running

=> Stored in stack



Unix Process memory space

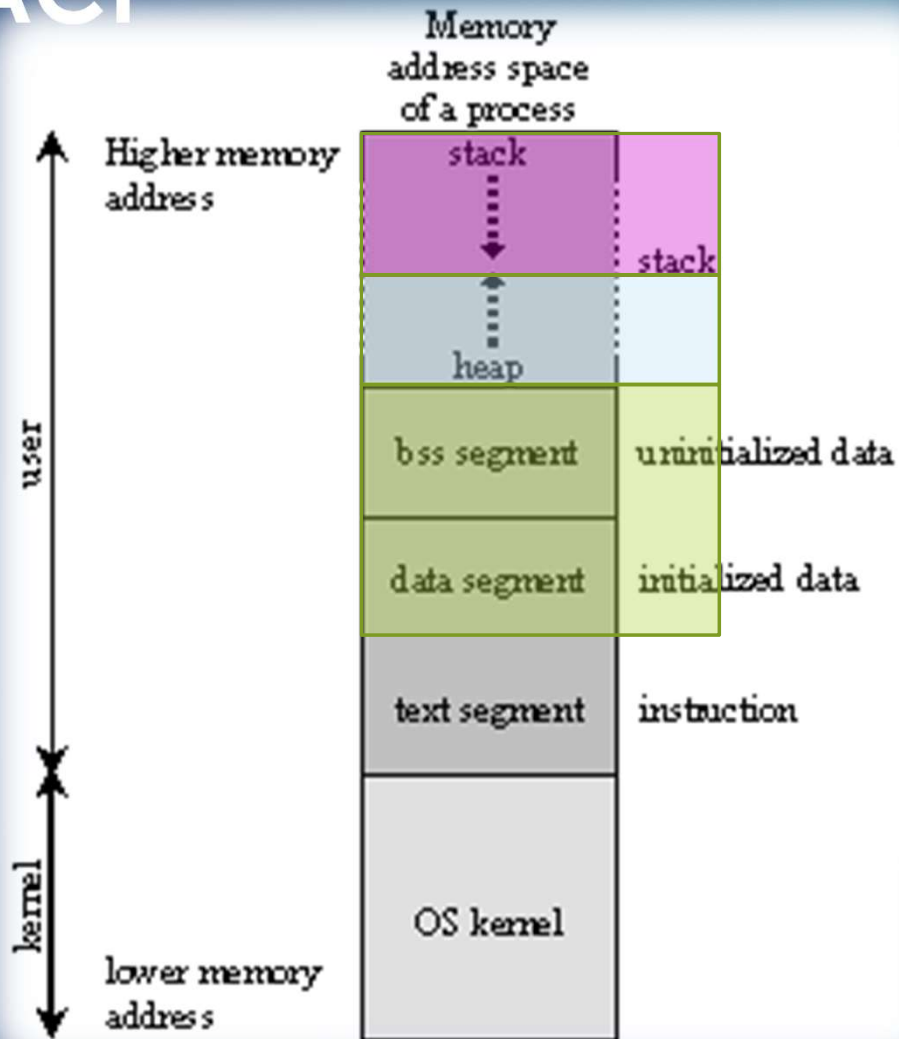
23

Local (dynamic) variables:
Function parameters,
Function local declaration,
Function return value,
Are stored in the **STACK**

Dynamic allocations (**malloc**)
Are stored in **HEAP**

Global (static) variables
Are stored in **data segment**

ACP



Unix Process memory space

24

```
Int globalVar;  
Int *globalPointer;
```

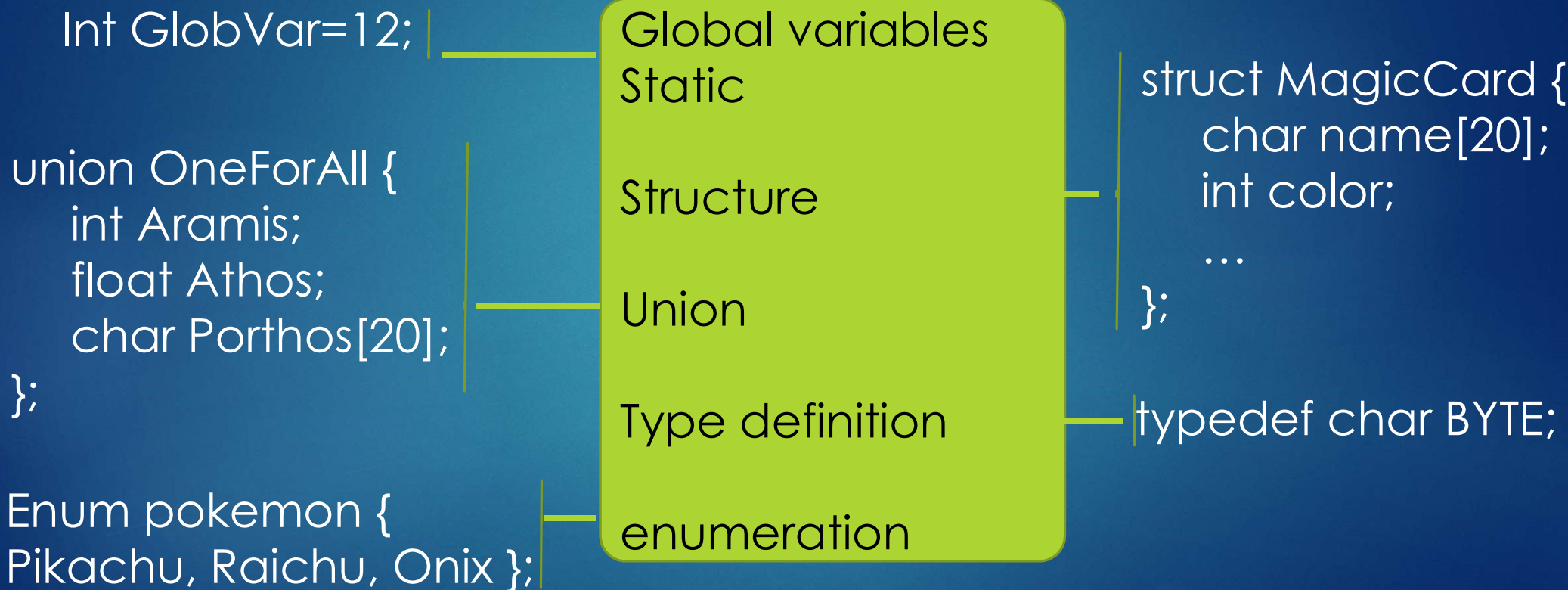
```
Int func(int parameter) {  
    int localVar;  
    int *localPointer;
```

```
    globalPointer=malloc(100);  
    localPointer = malloc(100);  
    return 1;  
}
```


ACP Global Structure of a C Program

Global declaration

25



Difference between global & local variables

26

```
Int global;  
Int func1() {  
    int loc1;  
    loc1=10;  
    global=20;  
    loc2=30;  
}  
Int func2() {  
    int loc2;  
    loc1=10;  
    global=20;  
    loc2=30;  
}
```

Error var loc2
not found

Error var loc1
not found

```
int test(int par) {  
    int testVal,retVal;
```

```
    testVal=par-1;  
    if(testVal==0) {  
        retVal=3;  
        return retVal;  
    }
```

```
    retVal= test(testVal)*testVal;  
    return retVal;  
}
```

```
Main() {  
    int enigma;  
    enigma= test(3);  
}
```

Stack & recursivity

28

```
int test(int par) {  
    int testVal,retVal;
```

```
    testVal=par-1;  
    if(testVal==0) {  
        retVal=3;  
        return retVal;  
    }
```

```
    retVal= test(testVal)*testVal;  
    return retVal;  
}
```

```
Main() {  
    int enigma;  
    enigma= test(3);  
}
```

Main() → stack
enigma=?

ACP

test(3)

Stack & recursivity

29

```
int test(int par) {  
    int testVal,retVal;  
  
    testVal=par-1;  
    if(testVal==0) {  
        retVal=3;  
        return retVal;  
    }  
  
    retVal= test(testVal)*testVal;  
    return retVal;  
}  
  
Main() {  
    int enigma;  
    enigma= test(3);  
}
```

Main()

test(3)

stack

enigma=?

par=3

retVal=?

testVal=?

Main()

ACP

Stack & recursivity

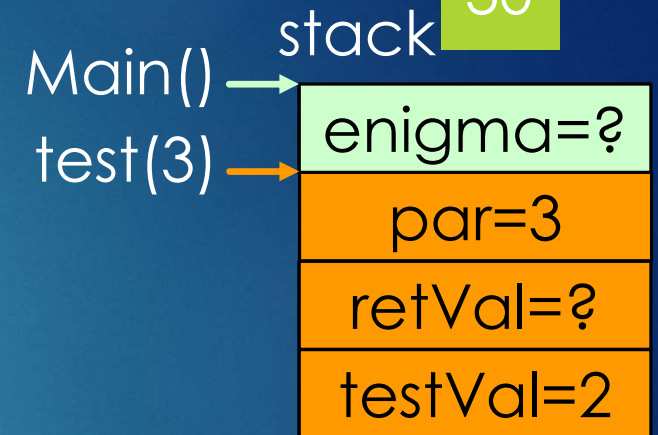
30

```
int test(int par) {  
    int testVal, retVal;
```

```
    testVal=par-1;  
    if(testVal==0) {  
        retVal=3;  
        return retVal;  
    }
```

```
    retVal= test(testVal)*testVal;  
    return retVal;  
}
```

```
Main() {  
    int enigma;  
    enigma= test(3);  
}
```



test(3)

Main()

ACR

test(2)

test(3)

Main()

Stack & recursivity

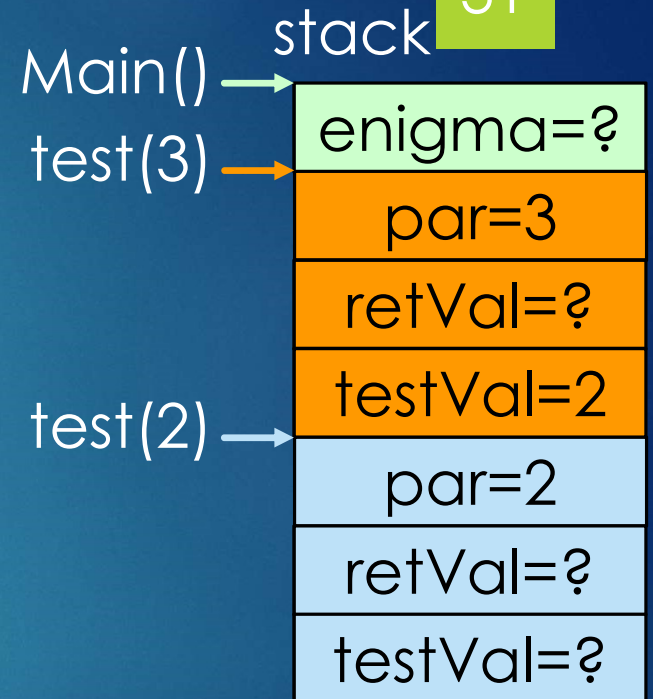
31

```
int test(int par) {  
    int testVal,retVal;
```

```
    testVal=par-1;  
    if(testVal==0) {  
        retVal=3;  
        return retVal;  
    }
```

```
    retVal= test(testVal)*testVal;  
    return retVal;  
}
```

```
Main() {  
    int enigma;  
    enigma= test(3);  
}
```



ACP

Stack & recursivity

32

```
int test(int par) {  
    int testVal,retVal;
```

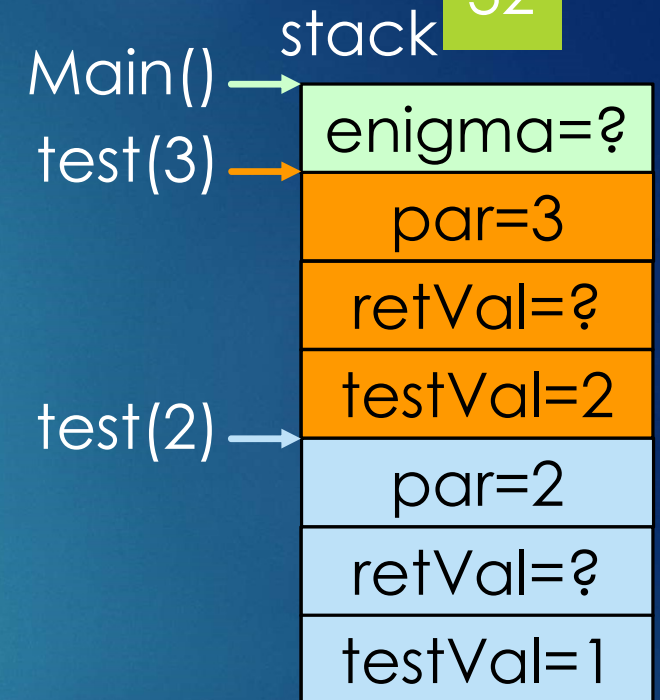
```
    testVal=par-1;  
    if(testVal==0) {  
        retVal=3;  
        return retVal;  
    }
```

```
    retVal= test(testVal)*testVal;  
    return retVal;  
}
```

```
Main() {  
    int enigma;  
    enigma= test(3);  
}
```

test(2)
test(3)

Main()

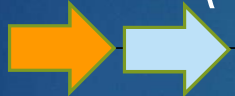


ACP



test(1)

test(2)



test(3)

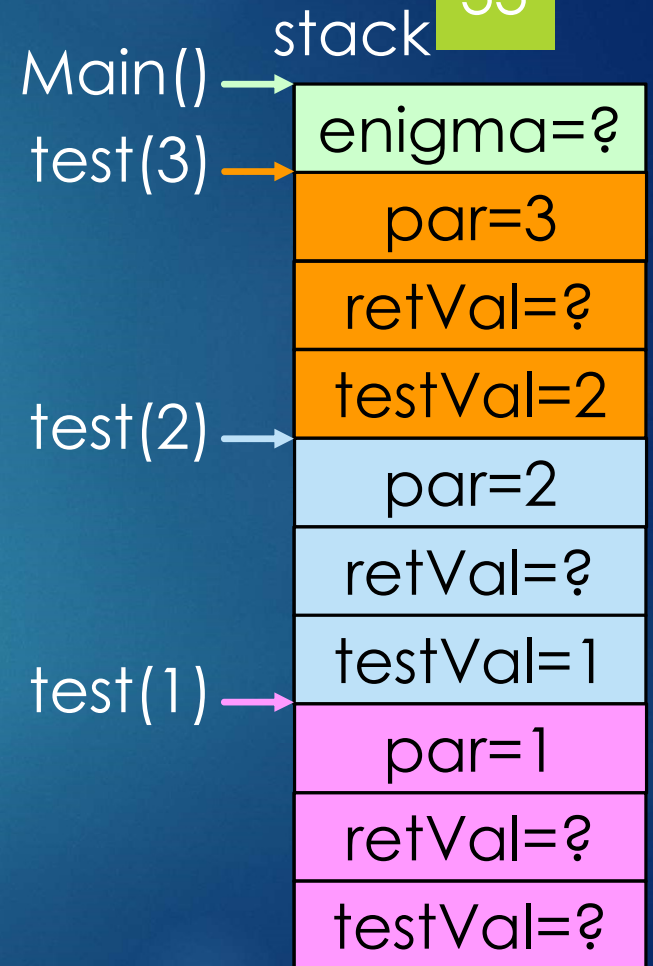
Main()



Stack & recursivity

33

```
int test(int par) {  
    int testVal,retVal;  
  
    testVal=par-1;  
    if(testVal==0) {  
        retVal=3;  
        return retVal;  
    }  
  
    retVal=test(testVal)*testVal;  
    return retVal;  
}  
  
Main() {  
    int enigma;  
    enigma=test(3);  
}
```



ACP

Stack & recursivity

34

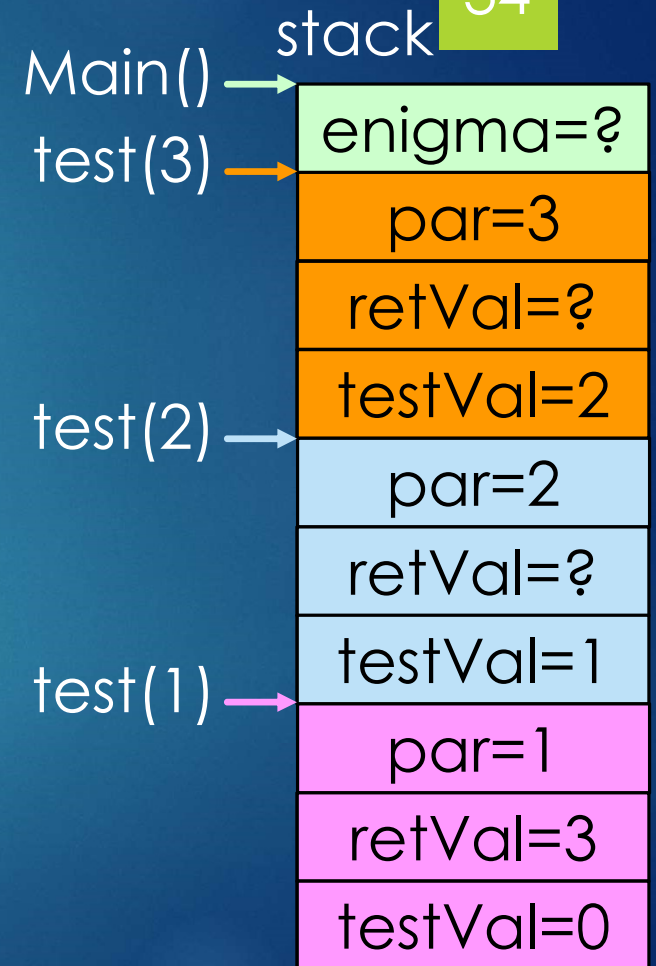
```
int test(int par) {  
    int testVal,retVal;  
  
    testVal=par-1;  
    if(testVal==0) {  
        retVal=3;  
        return retVal;  
    }  
  
    retVal=test(testVal)*testVal;  
    return retVal;  
}  
  
Main() {  
    int enigma;  
    enigma=test(3);  
}
```

test(1) →

test(2) →

test(3) →

Main() →



ACP

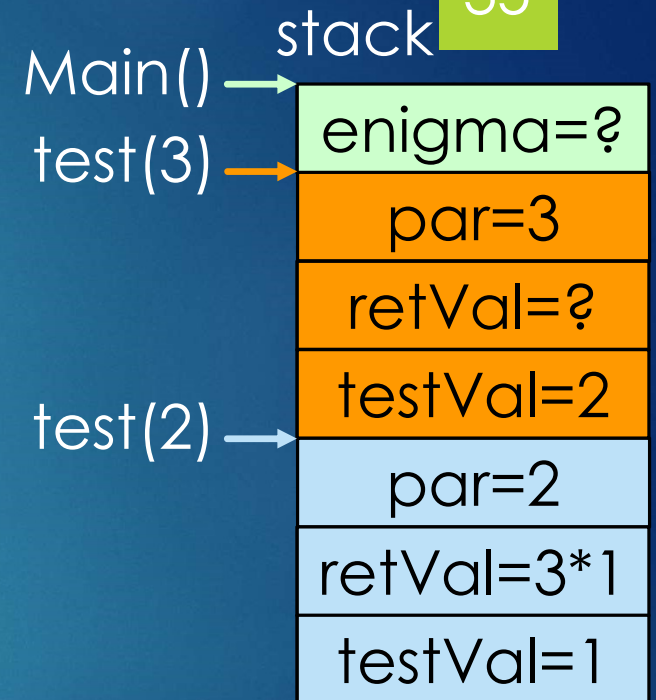
Stack & recursivity

35

```
int test(int par) {  
    int testVal,retVal;  
  
    testVal=par-1;  
    if(testVal==0) {  
        retVal=3;  
        return retVal;  
    }  
  
    retVal=test(testVal)*testVal;  
    return retVal;  
}  
  
Main() {  
    int enigma;  
    enigma=test(3);  
}
```

test(2)
test(3)

Main()

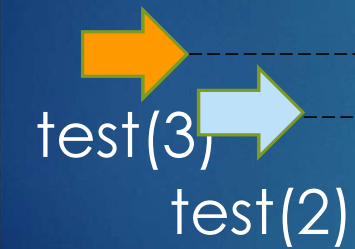
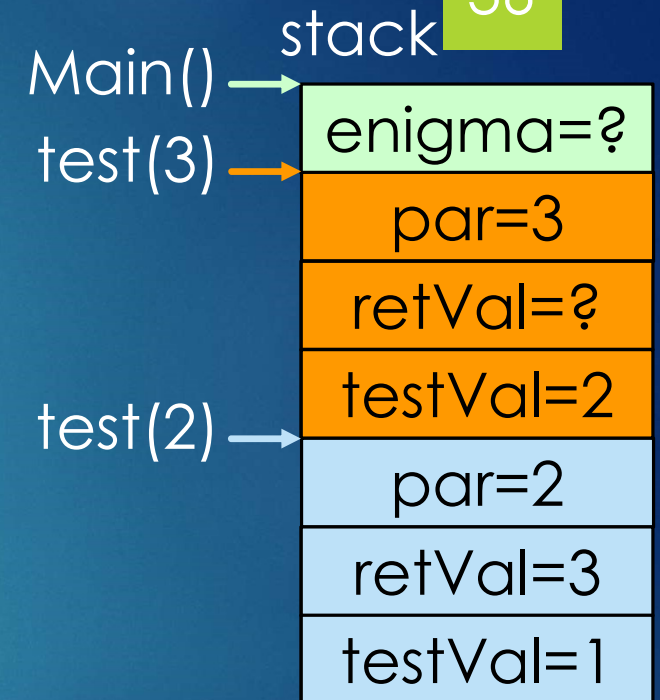


```
int test(int par) {  
    int testVal, retVal;
```

```
    testVal=par-1;  
    if(testVal==0) {  
        retVal=3;  
        return retVal;  
    }
```

```
    retVal=test(testVal)*testVal;  
    return retVal;  
}
```

```
Main() {  
    int enigma;  
    enigma=test(3);  
}
```

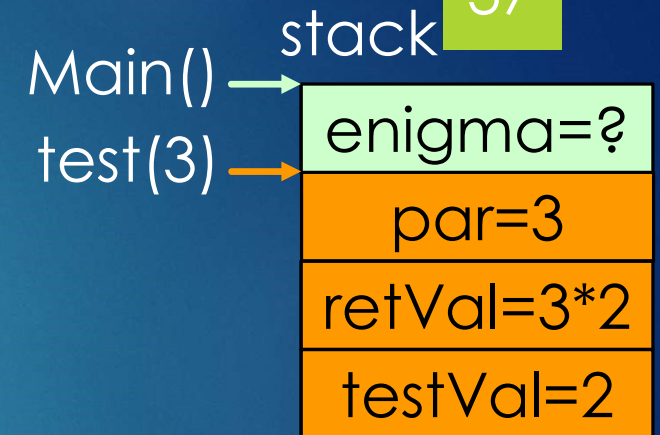



```
int test(int par) {  
    int testVal, retVal;
```

```
    testVal=par-1;  
    if(testVal==0) {  
        retVal=3;  
        return retVal;  
    }
```

```
    retVal=test(testVal)*testVal;  
    return retVal;  
}
```

```
Main() {  
    int enigma;  
    enigma=test(3);  
}
```



test(3)

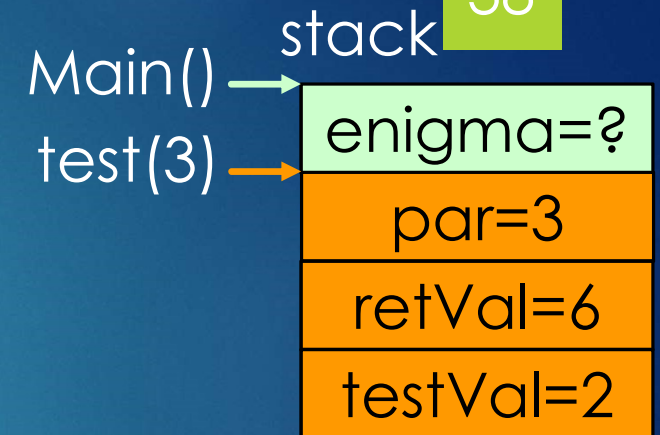
Main()

ACP

Stack & recursivity

38

```
int test(int par) {  
    int testVal,retVal;  
  
    testVal=par-1;  
    if(testVal==0) {  
        retVal=3;  
        return retVal;  
    }  
  
    retVal=test(testVal)*testVal;  
    return retVal;  
}  
  
Main() {  
    int enigma;  
    enigma=test(3);  
}
```

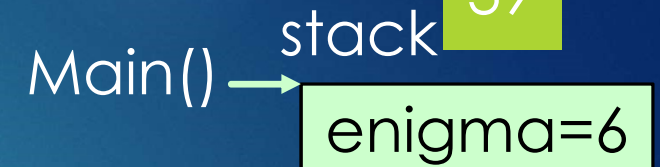


 `test(3)`

 `Main()`

```
int test(int par) {  
    int testVal,retVal;  
  
    testVal=par-1;  
    if(testVal==0) {  
        retVal=3;  
        return retVal;  
    }  
  
    retVal=test(testVal)*testVal;  
    return retVal;  
}
```

```
Main() {  
    int enigma;  
    enigma=test(3);  
}
```



Main()

```
Int rec(int par, int parRecLevel) {  
    int locRecLevel, retVal;  
  
    locRecLevel=parRecLevel++; // this one  
    locRecLevel=++parRecLevel; //or this one?  
    if(locRecLevel>=par) return par;  
    retVal=rec(par, locRecLevel)*locRecLevel;  
    return retVal;  
}  
  
Main() {  
    int v;  
    v=rec(3,0);  
}
```

rec(3,0) stack

par=3

parRecLevel=0

locRecLevel=1

retval

????

v=???

C

Variable types

Unsigned integers:

$$v = \sum_{i=0}^{N-1} u_i 2^i$$

2^3	2^2	2^1	2^0	
0	1	1	1	= 7
0	0	1	0	= 2
0	0	0	1	= 1
0	0	0	0	= 0
1	1	1	1	= 15
1	1	1	0	= 14
1	0	0	0	= 8

unsigned int : unsigned integer number on 8/16/32/64 bits depending on the CPU (size of the CPU register)

unsigned long : unsigned integer number on 64 bits

unsigned short : unsigned integer number on 16 bits

unsigned char : unsigned integer number on 8 bits

Signed integers:

$$v = -u_{N-1}2^{N-1} + \sum_{i=0}^{N-2} u_i 2^i$$

$-(2^3)$	2^2	2^1	2^0	
0	1	1	1	= 7
0	0	1	0	= 2
0	0	0	1	= 1
0	0	0	0	= 0
1	1	1	1	= -1
1	1	1	0	= -2
1	0	0	0	= -8

int : signed integer number on 8/16/32/64 bits depending on the CPU (size of the CPU register)

long : signed integer number on 64 bits

short : signed integer number on 16 bits

char : signed integer number on 8 bits

To initialise an integer:

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

```
void main() {  
    int spart=300;    // decimal value  
    spart=0x12C;      // hexadecimal value  
}
```

To display an integer:

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

```
void main() {  
    int tnotb=666;  
    printf("tnotb(decimal)=%d\n",tnotb);  
    printf("tnotb(hexadecimal)=%x\n",tnotb);  
}
```

Considering :

int are on 32 bits, char/8, short/16, long/64

What is the minimum value of unsigned char?

What is the maximum value of a short?

What is the minimum value of a long?

What is the maximum value of a unsigned long?

If you want to know the size of a variable use
sizeof

Example:

```
int varInt;
```

```
long varLong;
```

```
Printf("sizeof(varInt)=%d\n",sizeof(varInt));
```

```
Printf("sizeof(varLong)=%d\n",sizeof(varLong));
```


AND (&): intersection of two operands

$$0 \& 0 = 0$$

$$0 \& 1 = 0$$

$$1 \& 0 = 0$$

$$1 \& 1 = 1$$

Example on 4 bits operands:

$$\begin{array}{r} 0011 \\ \& 1010 \\ \hline = 0010 \end{array}$$

Masks can be used to reset some bits.

Ex: 0011 will reset the two first bits.

OR (|): union of two operands

0 | 0 = 0

0 | 1 = 1

1 | 0 = 1

1 | 1 = 1

Example on 4 bits operands:

0011

| 1010

= 1011

Masks can be used to set some bits.

Ex: 0011 will set the two last bits.

XOR (^): Exclusive OR of two operands

$$0 \wedge 0 = 0$$

$$0 \wedge 1 = 1$$

$$1 \wedge 0 = 1$$

$$1 \wedge 1 = 0$$

Example on 4 bits operands:

$$\begin{array}{r} 0011 \\ \wedge 1010 \\ \hline = 1001 \end{array}$$

Right shift (>>):

$op \gg shiftVal$: shift operand op of $shiftVal$ bits to the right

Arithmetic:

$op \gg shiftVal =$
 $op / 2^{shiftVal}$

Example on 4 bits operands:

$1011 \gg 1 = 0101$

$1011 \gg 2 = 0010$

$1011 \gg 3 = 0001$

$1011 \gg 4 = 0000$

Left shift (<<):

$op \ll shiftVal$: shift operand op of $shiftVal$ bits to the left

Example on 4 bits operands:

$1011 \ll 1 = 0110$

$1011 \ll 2 = 1100$

$1011 \ll 3 = 1000$

$1011 \ll 4 = 0000$

Arithmetic:

$op \ll shiftVal =$

$op \times 2^{shiftVal}$

Multiply two integers ($a \times b$) on 8 bits without using the multiply operator (*).

Hint:

Start by trying on fixed simple values.

ACP

char : 8 bits
integer
usually
used to
store ASCII
characters.

Standard
ASCII table
0-127

char

00	nul	10	dle	20	sp	30	0	40	@	50	P	60	`	70	p
01	soh	11	dc1	21	!	31	1	41	A	51	Q	61	a	71	q
02	stx	12	dc2	22	"	32	2	42	B	52	R	62	b	72	r
03	etx	13	dc3	23	#	33	3	43	C	53	S	63	c	73	s
04	eot	14	dc4	24	\$	34	4	44	D	54	T	64	d	74	t
05	enq	15	nak	25	%	35	5	45	E	55	U	65	e	75	u
06	ack	16	syn	26	&	36	6	46	F	56	V	66	f	76	v
07	bel	17	etb	27	'	37	7	47	G	57	W	67	g	77	w
08	bs	18	can	28	(38	8	48	H	58	X	68	h	78	x
09	ht	19	em	29)	39	9	49	I	59	Y	69	i	79	y
0a	nl	1a	sub	2a	*	3a	:	4a	J	5a	Z	6a	j	7a	z
0b	vt	1b	esc	2b	+	3b	;	4b	K	5b	[6b	k	7b	{
0c	np	1c	fs	2c	,	3c	<	4c	L	5c	W	6c	l	7c	
0d	cr	1d	gs	2d	-	3d	=	4d	M	5d]	6d	m	7d	}
0e	so	1e	rs	2e	.	3e	>	4e	N	5e	^	6e	n	7e	~
0f	si	1f	us	2f	/	3f	?	4f	O	5f	_	6f	o	7f	del

ACP

char

57

128	Ç	143	Å	158	℞	172	¼	186	∥	200	ℒ	214	∏	228	Σ	242	≥
129	ü	144	É	159	f	173	;	187	∥	201	ℒ	215	∏	229	σ	243	≤
130	é	145	æ	160	á	174	«	188	∥	202	ℒ	216	∏	230	μ	244	∫
131	â	146	Æ	161	í	175	»	189	∥	203	ℒ	217	∏	231	τ	245	∫
132	ä	147	ô	162	ó	176	⋮	190	∥	204	ℒ	218	∏	232	Φ	246	÷
133	à	148	ö	163	ú	177	■	191	∥	205	=	219	■	233	Θ	247	≈
134	å	149	ò	164	ñ	178	■	192	∥	206	∥	220	■	234	Ω	248	°
135	ç	150	û	165	Ñ	179	∥	193	∥	207	∥	221	■	235	δ	249	•
136	ê	151	ù	166	ª	180	∥	194	∥	208	∥	222	■	236	∞	250	•
137	ë	152	ÿ	167	º	181	∥	195	∥	209	∥	223	■	237	φ	251	√
138	è	153	Ö	168	¿	182	∥	196	∥	210	∥	224	α	238	ε	252	π
139	ï	154	Ü	169	¬	183	∥	197	∥	211	∥	225	β	239	∩	253	²
140	î	155	ç	170	¬	184	∥	198	∥	212	∥	226	Γ	240	≡	254	■
141	ì	156	£	171	½	185	∥	199	∥	213	∥	227	π	241	±	255	
142	Ä	157	¥														

Extended ASCII table

128-255

Translate in ASCII the String: "HAL9000"

How to transform this string in lowercase?

How to compare the alphabetical order of two strings : ex. "Dupond" & "Dupont"?

How to transform an ASCII number in "int"?

0xxxxxxx

0 1001000 = 72

110xxxxx 10xxxxxx

110 10010 10 010001 = 1169

11100000 101xxxxx 10xxxxxx

1110 0000 10 10101 10 11101 = 1275

11100001 10xxxxxx 10xxxxxx

1110001x 10xxxxxx 10xxxxxx

111001xx 10xxxxxx 10xxxxxx

111010xx 10xxxxxx 10xxxxxx

11101100 10xxxxxx 10xxxxxx

11101101 100xxxxx 10xxxxxx

1110111x 10xxxxxx 10xxxxxx

11110000 1001xxxx 10xxxxxx
10xxxxxx

11110000 101xxxxx 10xxxxxx
10xxxxxx

11110001 10xxxxxx 10xxxxxx
10xxxxxx

1111001x 10xxxxxx 10xxxxxx
10xxxxxx

11110100 1000xxxx 10xxxxxx
10xxxxxx

Utf8: Universal Character Set Transformation Format

$A=65=0x41=b(100\ 0001)=$ Utf8(b**0**100 0001)

$\sim = 128 = b(1000\ 0000) =$ Utf8(**11**00 0010 **10**00 0000)

$2048=0x800=b(1000\ 0000\ 0000)=$ Utf8(**111**0 0000 **10**10 0000 **10**00 0000)

ACP

char

61

To display a char :

```
#include <stdio.h>
void main() {
char myChar='A';
printf("myChar(decimal)=%d\n",myChar);
printf("myChar(hexadec.)=0x%x\n",myChar);
printf("myChar(character)=%c\n",myChar);
}
```

ACP

Utf8: Universal Character Set Transformation Format (ex.)

62

Decode this UTF8 encoded string
(hexadecimal code) :

0xE1 0x88 0xB4 0xE5 0x99 0xB8

Answer: 0x1234 0x5678

ACP

Utf8:Universal Character Set Transformation Format (ex.)

63

Encode these numbers in UTF8 String
(hexadecimal code) :
0XCAFE 0xBABE

Answer: 0xEC 0xAB 0xBE 0xEB 0xAA 0xBE

Floating Point IEEE754 standard :

$$(-1)^{\text{sign}} \times (1 + \text{mantissa}) \times 2^{(\text{exponent} - \text{bias})}$$

Mantissa (significand) = accuracy

Exponent = range

How to code 1? 2? 3? 0.25? 0?

float : real number on 32 bits :

1 sign bit, 8 exponent bits, 23 mantissa bits,
bias=127

double : real number on 64 bits

1 sign bit, 11 exponent bits, 52 mantissa bits,
bias=1023

Put 7558963.25 in a float variable.

Print it : `printf("num=%f\n",floatVar);`

What happens?

Why? (hint : find the mantissa of 7558963.25)

How to solve the problem?

```
// declaration
int myArray[11]; // array of 11 integers (from 0 to 10)
// use
myArray[0]=12;
myArray[i]=98;
// error
myArray[11]=90; // one of the most current error
```

```
// declaration with initialisation
int myArray[5]={10,20,30,40,50};
```

Create a static (global) array of ints.
Print the content of the array.

Create a dynamic (local) array of ints.
Print the content of the array.

Create a static array of 5 ints.
Read the 6th element. What happens?
Write the 6th element. What happens?

Strings are arrays of char:

```
char myString[100];
```

Initialisation:

```
myString[0]='A';
```

```
myString[1]='B';
```

```
myString[2]=(char)0;
```

```
myString[3]='C';
```

```
// to display a String  
Printf("string is %s\n",myString);
```

```
// The display will end when the  
// first 'null' (0) occurs. The result  
will be:
```

String is AB

Standard library `<string.h>` includes several functions to manipulate strings:

`#include <string.h>`

`int strlen(string);` // return the length of string (without the null character)

`int strcpy(stringDst, stringSrc);` // copy the stringSrc into stringDst.

`int strcmp(string1, string2);` // compare two strings.

- What happens if:
`char foo[3];`
`strcpy(foo, "weiner");`
`printf("foo=%s\n",foo);`
- Write a code which gives the index of the first difference.
`char string1 []="DUPOND";`
`char string2 []="DUPONT";`

```
// initialisation
```

```
int myMatrix[11][30]; // array of 11x30 integers
```

```
// use
```

```
myMatrix[0][0]=12;
```

```
myMatrix[i][j]=98;
```

```
// declaration with initialisation
```

```
int myMatrix[2][3]={ {1, 2, 3},  
                     {4, 5, 6} };
```

Try to find how is stored a multidimensional array.

Hint:

Create a $[3][2]$ matrix.

Initialise elements:

$[0][0]$, $[0][1]$, $[0][2]$, $[0][3]$, $[0][4]$, $[0][5]$

Visualise the content of:

$[0][0]$, $[0][1]$, $[1][0]$, $[1][1]$, $[2][0]$, $[2][1]$

Explain


```
// declaration:  
enum enumColor { RED, GREEN, BLUE};  
// => RED=0, GREEN=1, BLUE=2  
// variable declaration:  
enum enumColor myColor;  
// use  
myColor=BLUE;
```



```
// declaration:  
enum enumColor { RED, GREEN=4, BLUE};  
// => RED=0, GREEN=4, BLUE=5  
// variable declaration and initialisation:  
enum enumColor myColor=GREEN;
```

```
// enum declaration; variable declaration;  
// variable initialisation :
```

```
enum enumColor { RED, GREEN, BLUE} myColor=BLUE;
```

What happens when:

```
enum enumTest { FOO=3, BAR=2, MYSTERY };
```

ACP

Structure 1

78

```
// declaration  
struct date {  
    int day;  
    int month;  
    int year;  
};
```

```
// use  
yesterday.day=10;  
yesterday.month=1;  
yesterday.year=2099;
```

```
// declaration of one variable  
struct date yesterday;
```



```
// struct declaration; variable declaration;  
// variable initialisation  
struct date {  
    int day;  
    int month;  
    int year;  
} yesterday={ 10, 1, 2099};  
  
struct date foo[10];
```



```
// structure imbrication
struct event {
    struct date eDate;
    char name[32];
} testEvent= {{ 10, 1, 2099},
               {"JPO"} };

// use
testEvent.eDate.day=12;
```

ACP

Union

81

```
// Union declaration
union oneElement {
    float floatVal;
    int intVal;
    char charVal[10];
};
```

```
union oneElement foo;
foo.floatVal=3.14;
```

Foo will be able to take only one of the Three available element:

foo.floatVal OR
foo.intVal OR
foo.charVal

The size of foo will be equal to the longer field. In this case the size of foo will be 10 bytes.

&var1	Value(var1) Value(*pt1)
&pt1	Value(pt1)=&var1

```
int var1;  
int *pt1; // declare pt1 as a pointer on ints  
  
pt1=&var1; // put the address of var1 in pt1  
*pt1=12;   // put 12 in the memory pointed  
           // by pt1. As pt1 "points" on var1  
           // it is the same as var1=12;  
printf("var1=%d\n",var1); // will give var1=12  
  
var1=6;    // modify the content of var1  
printf("*pt1=%d\n",*pt1); // will give *pt1=6
```

```
int arrayInt[5]={1,2,3,4,5};  
int *ptInt; // declare ptInt as a pointer on ints  
char arrayChar[5]={"JPO"};  
char *ptChar;
```

```
ptInt=&arrayInt[0];    // same as ptInt=&arrayInt  
ptChar=&arrayChar[0];  
printf("%d %c\n",*ptInt, *ptChar); // output : 1 J  
ptInt=ptInt+1; ptChar=ptChar+1;  
printf("%d %c\n",*ptInt, *ptChar); // output : 2 P
```

Operation on pointers depends on their types:

+1 on an int will be +4 bytes

+1 on a char will be +1 byte

It is possible to use operator `[]` (array) on pointers:

```
int arrayInt[5]={1,2,3,4,5};  
int *ptInt; // declare ptInt as a pointer on ints  
char arrayChar[5]={"JPO"};  
char *ptChar;  
  
ptInt=&arrayInt[0]; // same as ptInt=&arrayInt  
ptChar=&arrayChar[0];  
printf("%d %c\n",ptInt[0], ptChar[0]); // output : 1 J  
printf("%d %c\n",ptInt[1], ptChar[1]); // output : 2 P
```

`pt[n]`
is equivalent to
`*(pt+n)`

Big endian:

0x12345678

0x12

@n

0x34

@n+1

0x56

@n+2

0x78

@n+3

Little endian:

0x78

@n

0x56

@n+1

0x34

@n+2

0x12

@n+3

```
int var=0x12345678;  
char *pt;
```

```
pt=(char *)&var;  
printf("v1=%x v2=%x v3=%x v4=%x\n",pt[0], pt[1], pt[2], pt[3]);
```

=> Output???

Find the coding of <type> on your machine:
Long (is it little or big endian?)
float, double (where is the
sign/mantissa/exponent?)

Is it possible to visualize the stack?
(hint: use recurse function)

```
#include <malloc.h>
```

```
void *malloc(int size);
```

Allocate *size* bytes in heap space and return a pointer to this reserved space.

Example:

```
int *pt;
```

```
pt=(int *)malloc(sizeof(int));
```

Always cast to
the right type

Use sizeof to get
the right size


```
#include <malloc.h>
```

```
void free(void *foo);
```

Deallocate (free) the space reserved in heap space pointed by *foo*.

Example:

```
int *pt;
```

```
pt=(int *)malloc(sizeof(int));
```

```
free(pt);
```

Allocate an array of 32 characters.

Initialise it with the String “Run to the hills”

Display it with a printf

Free it

```
struct student {  
    char *name;  
    int *notes;  
}
```

```
struct student *singleStudent;  
struct student studentStatArray[100];
```

```
int main() {  
    singleStudent=(struct student *)malloc(sizeof(struct student));  
    singleStudent->name=(char *)malloc(100);  
    singleStudent->notes=(int *)malloc(sizeof(int)*10);  
  
    studentStatArray[0].name=(char *)malloc(100);  
    studentStatArray[0].notes=(int *)malloc(sizeof(int)*10);  
}
```

```
struct student {  
    char *name;  
    int *notes;  
}
```

```
struct student *studentDynArray;
```

```
int main() {  
    studentDynArray=(struct student *)malloc(sizeof(struct student)*10);  
  
    studentDynArray[0].name=(char *)malloc(100);  
    studentDynArray[0].notes=(int *)malloc(sizeof(int)*10);  
}
```

Pointer used as an array


```
struct chainList {  
    int val;  
    struct chainList *next;  
};
```

```
int main() {  
    struct chainList *startList, *lastElm, *newElm;
```

```
    startList=(struct chainList *)malloc(sizeof(struct chainList));  
    startList->val=0;  
    lastElm=startList;  
    newElm=(struct chainList *)malloc(sizeof(struct chainList));  
    newElm->val=1;  
    newElm->next=NULL;  
    lastElm->next=newElm;  
    lastElm=newElm;
```

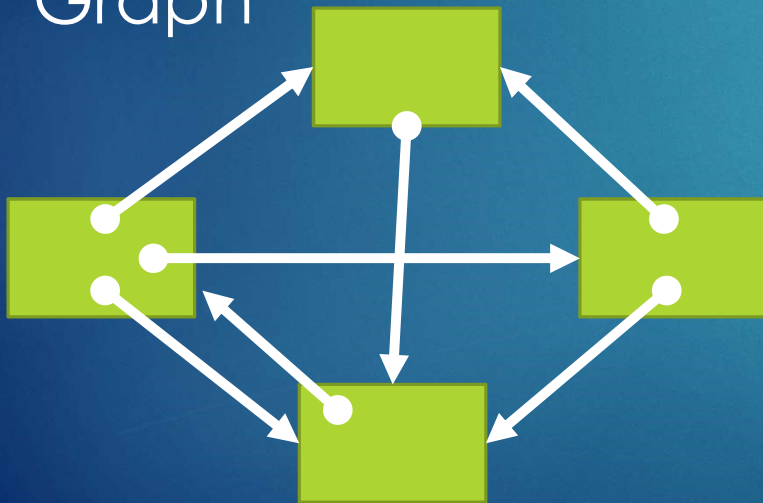
```
}
```



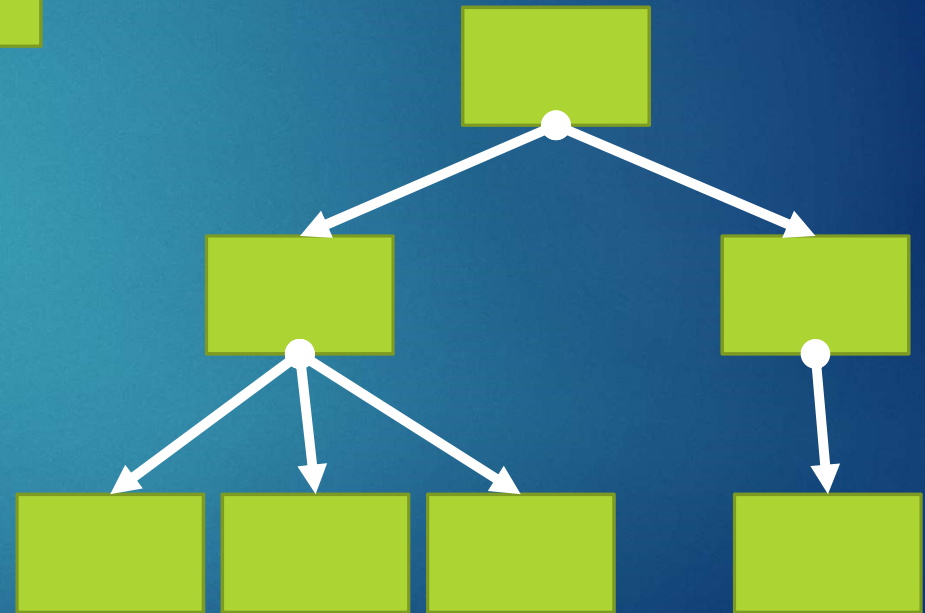
2 ways Chained list



Graph



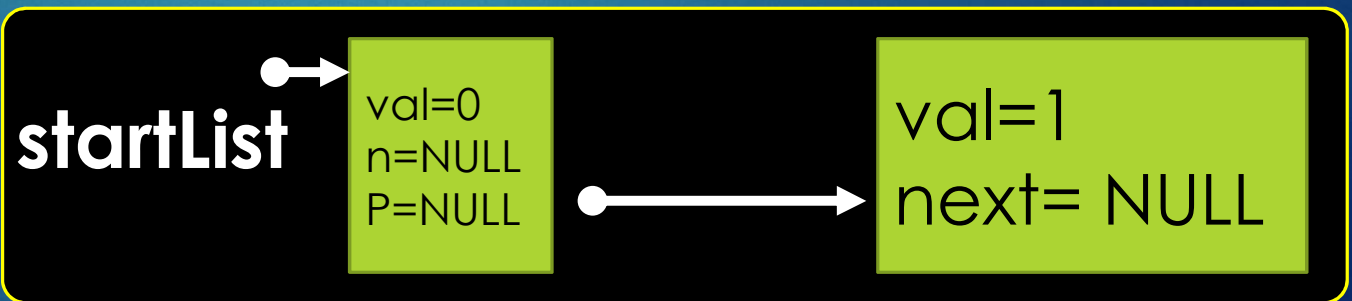
Tree



```
struct chainList {  
    int val;  
    struct chainList *n, *p; // n: next, p: prev  
};  
int main() {  
    struct chainList *startList, *lastElm, *newElm;
```

```
    startList=(struct chainList *)malloc(sizeof(struct chainList));  
    startList->val=0; startList->n=NULL; startList->p=NULL;  
    lastElm=startList;  
    newElm=(struct chainList *)malloc(sizeof(struct chainList));  
    newElm->val=1;  
    newElm->n=NULL;  
    lastElm->n=newElm;  
    lastElm=newElm;
```

```
}
```



```
typedef struct chainList {  
    int val;  
    struct chainList *n, *p; // n: next, p: prev  
} t_chain;  
int main() {  
    1 t_chain *start, *last, *newElm;  
  
    start=(t_chain *)malloc(sizeof(t_chain));  
    start->val=0; start->n=NULL; start->p=NULL;  
    last=start;
```

```
    newElm=(t_chain *)malloc(sizeof(t_chain));  
    newElm->val=1; newElm->n=NULL; newElm->p=NULL;  
    last->n=newElm;  
    newElm->p=last;  
    last=newElm;  
    newElm=(t_chain *)malloc(sizeof(t_chain));  
    newElm->val=2; newElm->n=NULL; newElm->p=NULL;  
    last->n=newElm;  
    newElm->p=last;  
    last=newElm;  
}
```

start
●


```
typedef struct chainList {  
    int val;  
    struct chainList *n, *p; // n: next, p: prev  
} t_chain;  
int main() {  
    t_chain *start, *last, *newElm;  
  
    2 start=(t_chain *)malloc(sizeof(t_chain));  
    start->val=0; start->n=NULL; start->p=NULL;  
    last=start;
```

```
    newElm=(t_chain *)malloc(sizeof(t_chain));  
    newElm->val=1; newElm->n=NULL; newElm->p=NULL;  
    last->n=newElm;  
    newElm->p=last;  
    last=newElm;  
    newElm=(t_chain *)malloc(sizeof(t_chain));  
    newElm->val=2; newElm->n=NULL; newElm->p=NULL;  
    last->n=newElm;  
    newElm->p=last;  
    last=newElm;
```

}

start



val=?
n=?
p=?

```
typedef struct chainList {
    int val;
    struct chainList *n, *p; // n: next, p: prev
} t_chain;
int main() {
    t_chain *start, *last, *newElm;

    start=(t_chain *)malloc(sizeof(t_chain));
    start->val=0; start->n=NULL; start->p=NULL;
    last=start;
```

3

```
newElm=(t_chain *)malloc(sizeof(t_chain));
newElm->val=1; newElm->n=NULL; newElm->p=NULL;
last->n=newElm;
newElm->p=last;
last=newElm;
newElm=(t_chain *)malloc(sizeof(t_chain));
newElm->val=2; newElm->n=NULL; newElm->p=NULL;
last->n=newElm;
newElm->p=last;
last=newElm;
}
```

start

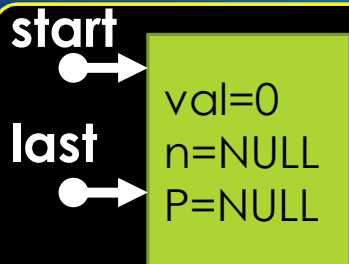


```
val=0
n=NULL
P=NULL
```

```
typedef struct chainList {  
    int val;  
    struct chainList *n, *p; // n: next, p: prev  
} t_chain;  
int main() {  
    t_chain *start, *last, *newElm;  
  
    start=(t_chain *)malloc(sizeof(t_chain));  
    start->val=0; start->n=NULL; start->p=NULL;  
    4 last=start;
```

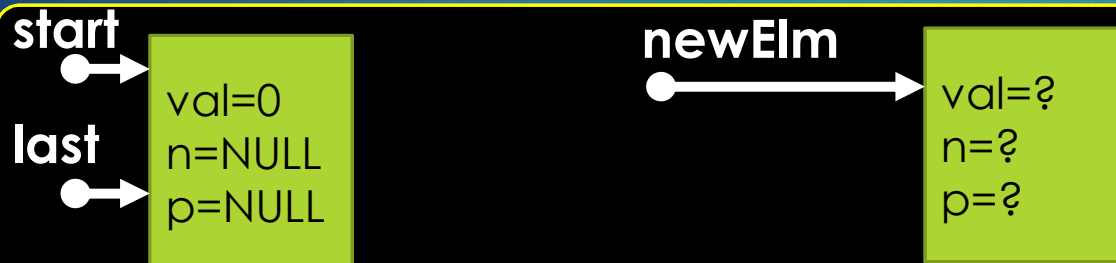
```
    newElm=(t_chain *)malloc(sizeof(t_chain));  
    newElm->val=1; newElm->n=NULL; newElm->p=NULL;  
    last->n=newElm;  
    newElm->p=last;  
    last=newElm;  
    newElm=(t_chain *)malloc(sizeof(t_chain));  
    newElm->val=2; newElm->n=NULL; newElm->p=NULL;  
    last->n=newElm;  
    newElm->p=last;  
    last=newElm;
```

}



```
typedef struct chainList {  
    int val;  
    struct chainList *n, *p; // n: next, p: prev  
} t_chain;  
int main() {  
    t_chain *start, *last, *newElm;  
  
    start=(t_chain *)malloc(sizeof(t_chain));  
    start->val=0; start->n=NULL; start->p=NULL;  
    last=start;
```

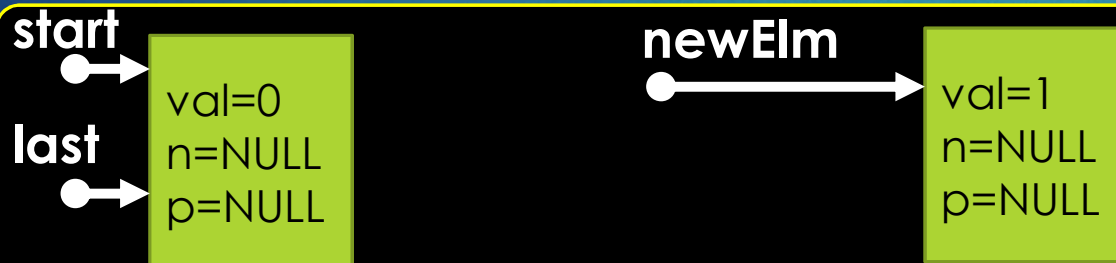
```
5 newElm=(t_chain *)malloc(sizeof(t_chain));  
  newElm->val=1; newElm->n=NULL; newElm->p=NULL;  
  last->n=newElm;  
  newElm->p=last;  
  last=newElm;  
  newElm=(t_chain *)malloc(sizeof(t_chain));  
  newElm->val=2; newElm->n=NULL; newElm->p=NULL;  
  last->n=newElm;  
  newElm->p=last;  
  last=newElm;  
}
```




```
typedef struct chainList {  
    int val;  
    struct chainList *n, *p; // n: next, p: prev  
} t_chain;  
int main() {  
    t_chain *start, *last, *newElm;  
  
    start=(t_chain *)malloc(sizeof(t_chain));  
    start->val=0; start->n=NULL; start->p=NULL;  
    last=start;
```

6

```
    newElm=(t_chain *)malloc(sizeof(t_chain));  
    newElm->val=1; newElm->n=NULL; newElm->p=NULL;  
    last->n=newElm;  
    newElm->p=last;  
    last=newElm;  
    newElm=(t_chain *)malloc(sizeof(t_chain));  
    newElm->val=2; newElm->n=NULL; newElm->p=NULL;  
    last->n=newElm;  
    newElm->p=last;  
    last=newElm;  
}
```



```
typedef struct chainList {  
    int val;  
    struct chainList *n, *p; // n: next, p: prev  
} t_chain;  
int main() {  
    t_chain *start, *last, *newElm;  
  
    start=(t_chain *)malloc(sizeof(t_chain));  
    start->val=0; start->n=NULL; start->p=NULL;  
    last=start;
```

```
        newElm=(t_chain *)malloc(sizeof(t_chain));  
        newElm->val=1; newElm->n=NULL; newElm->p=NULL;  
        last->n=newElm;  
        newElm->p=last;  
        last=newElm;  
        newElm=(t_chain *)malloc(sizeof(t_chain));  
        newElm->val=2; newElm->n=NULL; newElm->p=NULL;  
        last->n=newElm;  
        newElm->p=last;  
        last=newElm;  
    }
```



```

typedef struct chainList {
    int val;
    struct chainList *n, *p; // n: next, p: prev
} t_chain;
int main() {
    t_chain *start, *last, *newElm;

    start=(t_chain *)malloc(sizeof(t_chain));
    start->val=0; start->n=NULL; start->p=NULL;
    last=start;

```

8

```

    newElm=(t_chain *)malloc(sizeof(t_chain));
    newElm->val=1; newElm->n=NULL; newElm->p=NULL;
    last->n=newElm;
    newElm->p=last;
    last=newElm;
    newElm=(t_chain *)malloc(sizeof(t_chain));
    newElm->val=2; newElm->n=NULL; newElm->p=NULL;
    last->n=newElm;
    newElm->p=last;
    last=newElm;
}

```



```
typedef struct chainList {  
    int val;  
    struct chainList *n, *p; // n: next, p: prev  
} t_chain;  
int main() {  
    t_chain *start, *last, *newElm;  
  
    start=(t_chain *)malloc(sizeof(t_chain));  
    start->val=0; start->n=NULL; start->p=NULL;  
    last=start;
```

```
    newElm=(t_chain *)malloc(sizeof(t_chain));  
    newElm->val=1; newElm->n=NULL; newElm->p=NULL;  
    last->n=newElm;  
    newElm->p=last;  
    last=newElm;  
    newElm=(t_chain *)malloc(sizeof(t_chain));  
    newElm->val=2; newElm->n=NULL; newElm->p=NULL;  
    last->n=newElm;  
    newElm->p=last;  
    last=newElm;  
}
```




```

typedef struct chainList {
    int val;
    struct chainList *n, *p; // n: next, p: prev
} t_chain;
int main() {
    t_chain *start, *last, *newElm;

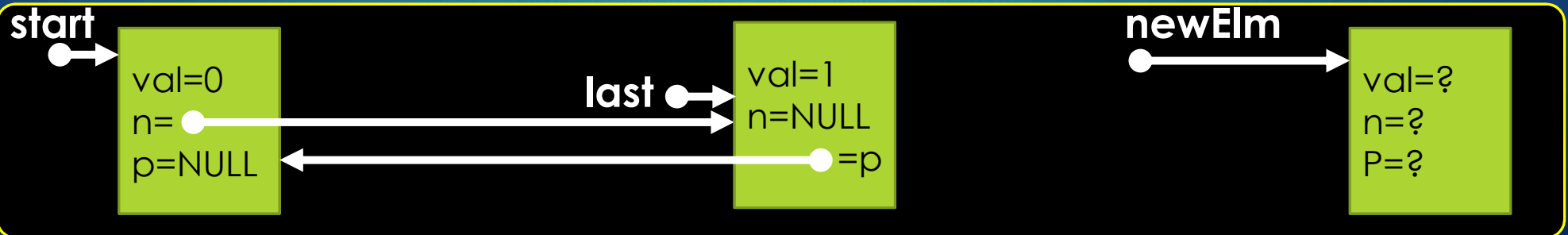
    start=(t_chain *)malloc(sizeof(t_chain));
    start->val=0; start->n=NULL; start->p=NULL;
    last=start;

```

```

    newElm=(t_chain *)malloc(sizeof(t_chain));
    newElm->val=1; newElm->n=NULL; newElm->p=NULL;
    last->n=newElm;
    newElm->p=last;
    last=newElm;
10 newElm=(t_chain *)malloc(sizeof(t_chain));
    newElm->val=2; newElm->n=NULL; newElm->p=NULL;
    last->n=newElm;
    newElm->p=last;
    last=newElm;
}

```



```

typedef struct chainList {
    int val;
    struct chainList *n, *p; // n: next, p: prev
} t_chain;
int main() {
    t_chain *start, *last, *newElm;

    start=(t_chain *)malloc(sizeof(t_chain));
    start->val=0; start->n=NULL; start->p=NULL;
    last=start;

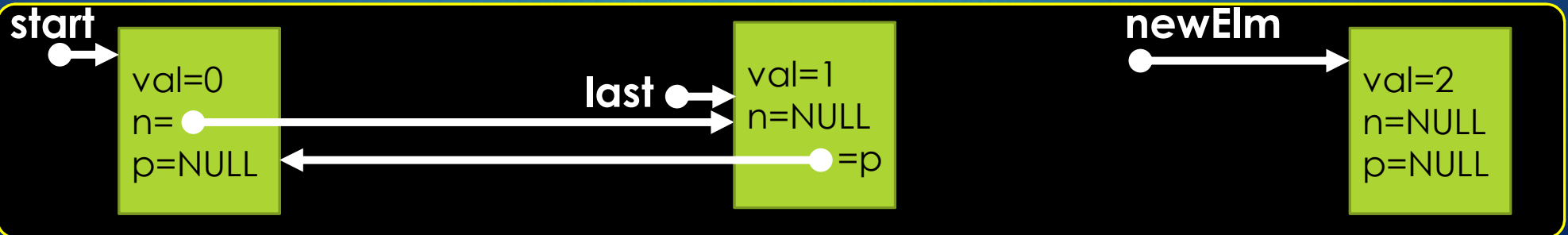
```

```

    newElm=(t_chain *)malloc(sizeof(t_chain));
    newElm->val=1; newElm->n=NULL; newElm->p=NULL;
    last->n=newElm;
    newElm->p=last;
    last=newElm;
    newElm=(t_chain *)malloc(sizeof(t_chain));
    newElm->val=2; newElm->n=NULL; newElm->p=NULL;
    last->n=newElm;
    newElm->p=last;
    last=newElm;
}

```

11



```

typedef struct chainList {
    int val;
    struct chainList *n, *p; // n: next, p: prev
} t_chain;
int main() {
    t_chain *start, *last, *newElm;

    start=(t_chain *)malloc(sizeof(t_chain));
    start->val=0; start->n=NULL; start->p=NULL;
    last=start;

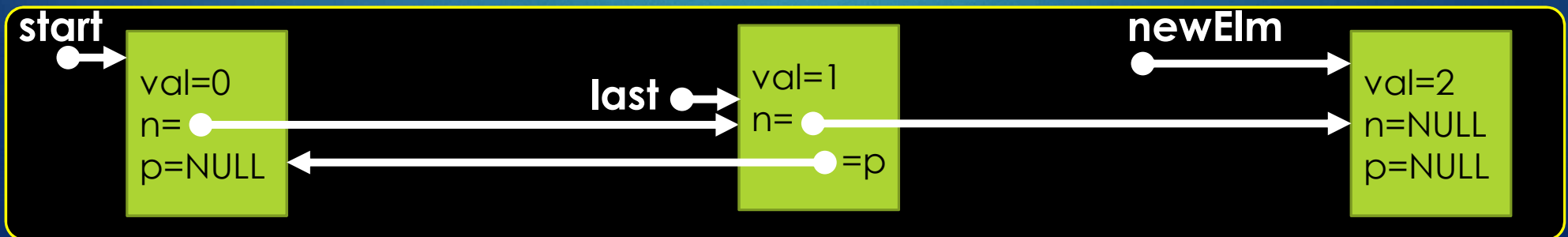
```

```

    newElm=(t_chain *)malloc(sizeof(t_chain));
    newElm->val=1; newElm->n=NULL; newElm->p=NULL;
    last->n=newElm;
    newElm->p=last;
    last=newElm;
    newElm=(t_chain *)malloc(sizeof(t_chain));
    newElm->val=2; newElm->n=NULL; newElm->p=NULL;
    last->n=newElm;
    newElm->p=last;
    last=newElm;
}

```

12



```

Typedef struct chainList {
    int val;
    struct chainList *n, *p; // n: next, p: prev
} t_chain;
int main() {
    t_chain *start, *last, *newElm;

    start=(t_chain *)malloc(sizeof(t_chain));
    start->val=0; start->n=NULL; start->p=NULL;
    last=start;

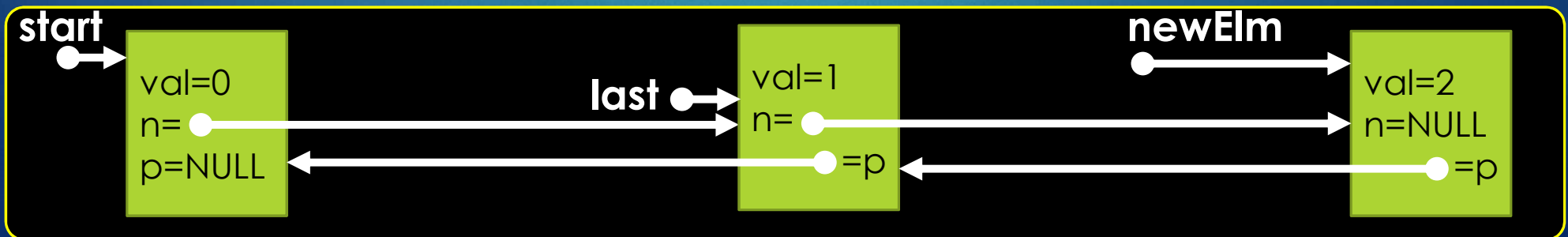
```

```

    newElm=(t_chain *)malloc(sizeof(t_chain));
    newElm->val=1; newElm->n=NULL; newElm->p=NULL;
    last->n=newElm;
    newElm->p=last;
    last=newElm;
    newElm=(t_chain *)malloc(sizeof(t_chain));
    newElm->val=2; newElm->n=NULL; newElm->p=NULL;
    last->n=newElm;
    newElm->p=last;
    last=newElm;
}

```

13




```

typedef struct chainList {
    int val;
    struct chainList *n, *p; // n: next, p: prev
} t_chain;

int main() {
    t_chain *start, *last, *newElm;

    start=(t_chain *)malloc(sizeof(t_chain));
    start->val=0; start->n=NULL; start->p=NULL;
    last=start;

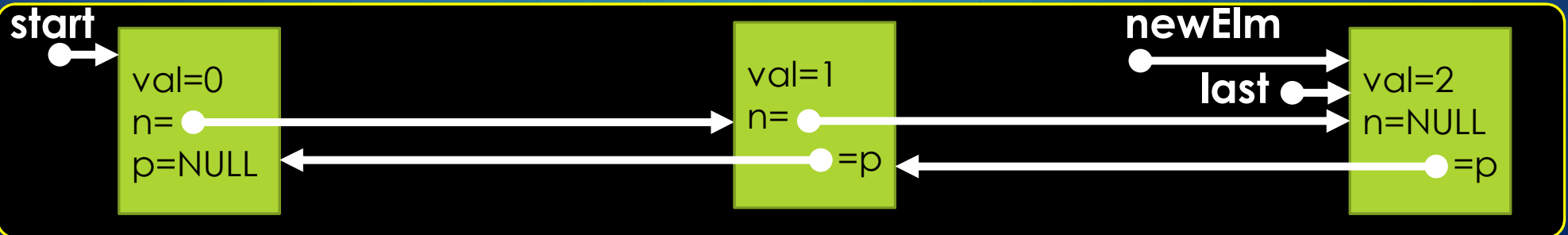
```

```

    newElm=(t_chain *)malloc(sizeof(t_chain));
    newElm->val=1; newElm->n=NULL; newElm->p=NULL;
    last->n=newElm;
    newElm->p=last;
    last=newElm;
    newElm=(t_chain *)malloc(sizeof(t_chain));
    newElm->val=2; newElm->n=NULL; newElm->p=NULL;
    last->n=newElm;
    newElm->p=last;
    last=newElm;
}

```

14

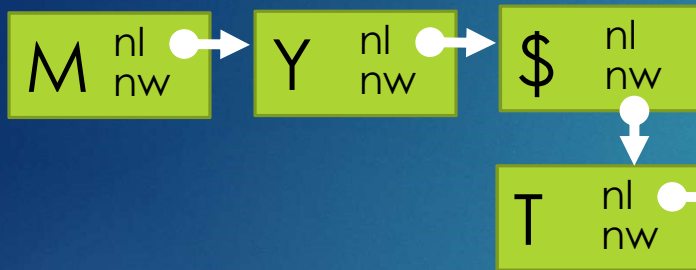


- Write a function which starts from “start” and insert an element “42” between the elements “1” and “2”.
- **MANDATORY** : As in the example, draw each step of your code.

- Write a function which creates a “2 ways chain list” of n elements : each element will store a number. The first element will have number one, the second number 2, etc. This function return the first element of the list.
- Elm *createChainList(int sizeList)
- Read list from first to last and last to first
- Void readFirst2Last(Elm *firstElm)

- Write a function which inserts an element at position n from the first element. This function returns the position of the first element of the list
- `Elm *insertElm(int position, Elm *elm2Insert)`

- Write a completion dictionary

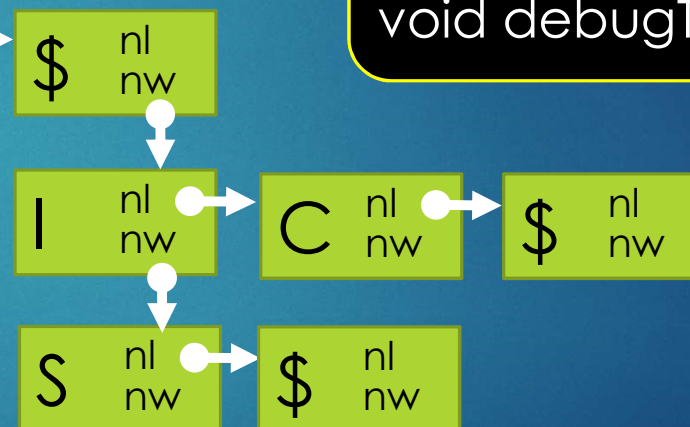


Create:
`int addWord(char *string)`
`void debugTree();`

```
addWord("MY");  
addWord("MYTH");  
addWord("MYTHIC");  
addWord("MYTHS");  
debugTree() output:
```

```
MY  
MYTH  
MYTHIC  
MYTHS
```

Order can be
different



```
int originalFunc(int x, int y) {  
    printf("x=%d, y=%d\n", x, y);  
    return 0;  
}  
int main() {  
    int (*ptFunc)(int x, int y);  
    ptFunc=originalFunc;  
    ptFunc(2, 3);  
    return 0;  
}
```

Object Oriented Approach

- Write a single chained list of heterogeneous elements: rectangle(left, top, width, height), circle(radius), triangle(x1,y1,x2,y2,x3,y3).
- Each element will have a function “display” which will write to the screen the parameters of the matching element.

Pointers on functions (ex.)

Object Oriented Approach : hints

```
struct object {  
    struct object *next;  
    void (*display)(struct object *object);  
};
```

```
struct rectangle {  
    struct object *next;  
    void (*display)(struct object *pt);  
    int left, top, width, height;  
}
```

```
struct circle {  
    struct object *next;  
    void (*display)(struct object *pt);  
    int radius;  
};
```

```
struct triangle {  
    struct object *next;  
    void (*display)(struct object *pt);  
    int x1, y1, x2, y2, x3, y3;  
}
```

```
struct circle *createCircle(int radius);  
struct rectangle *createRectangle(int left, int top, int width, int height);  
struct triangle *createTriangle(int x1, int y1, int x2, int y2, int x3, int y3);
```

```
void displayTriangle(struct triangle *pt);  
void displayCircle(struct circle *pt);  
void displayRectangle(struct rectangle *pt);
```

```
void addObject(struct object *pt);  
void displayList();
```


By value :

```
void resetAB(int a, int b) {  
    a=0; b=0;  
}
```

```
int a=1,b=2;  
resetAB(a,b);  
Printf("a=%d b=%d\n",a,b);  
=> Output : a=1 b=2
```


By reference (or address):

```
void resetAB(int *a, int *b) {  
    *a=0; *b=0;  
}
```

```
int a=1,b=2;  
resetAB(&a, &b);  
Printf("a=%d b=%d\n",a,b);  
=> Output : a=0 b=0
```

open()

```
#include <fcntl.h>
```

```
int open( char *filename, int access, int permission );
```

The available access modes are

O_RDONLY O_WRONLY O_RDWR O_APPEND O_BINARY O_TEXT

The permissions are

S_IWRITE S_IREAD S_IWRITE | S_IREAD

The *open()* function returns an integer value, which is used to refer to the file. If un- successful, it returns -1, and sets the global variable *errno* to indicate the error type.

read()

#include <fcntl.h>

*int read(int handle, void *buffer, int nbytes);*

The *read()* function attempts to read *nbytes* from the file associated with *handle*, and places the characters read into *buffer*. If the file is opened using *O_TEXT*, it removes carriage returns and detects the end of the file.

The function returns the number of bytes read. On end-of-file, 0 is returned, on error it returns -1, setting *errno* to indicate the type of error that occurred.

write()

```
#include <fcntl.h>
```

```
int write( int handle, void *buffer, int nbyte );
```

The *write()* function attempts to write *nbytes* from *buffer* to the file associated with *handle*. On text files, it expands each LF to a CR/LF.

The function returns the number of bytes written to the file. A return value of -1 indicates an error, with *errno* set appropriately.

close()

```
#include <fcntl.h>  
int close( int handle );
```

The `close()` function closes the file associated with `handle`. The function returns 0 if successful, -1 to indicate an error, with `errno` set appropriately.


```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

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

int main() {

    int fh;
    char str[]="1234567890ABCDEFGHIJKLMOP";
    int stat;

    fh = open("abc.txt", O_CREAT | O_WRONLY,
S_IRUSR);
    printf ("File handle %d\n",fh);
    stat = write(fh,str,strlen(str));
    if(stat<=0) printf("error while writing
data\n");
}
```

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#include <stdlib.h>
#include <stdio.h>

int main() {

    int fh;
    char buffer[65];
    int gotten;

    fh = open("abc.txt", O_RDONLY, S_IRUSR);
    printf ("File handle %d\n", fh);
    while (gotten = read(fh, buffer, 64)) {
        buffer[gotten] = '\0';
        printf ("*****%s", buffer);
    }
}
```

ACP File manipulation & pointers (ex.)

125

- Open the file student.txt
- Create an array of all student elements.
- Sort students by alphabetical order
- Write a file student_order.txt with all students sorted by alphabetical order.

ACP

Qsort algorithm

126

Bias=3

3	1	4	1	5	9	2	6	5	3
---	---	---	---	---	---	---	---	---	---

3	1	3	1	2	9	5	6	5	4
---	---	---	---	---	---	---	---	---	---

Bias=3

3	1	3	1	2	9	5	6	5	4
---	---	---	---	---	---	---	---	---	---

Bias=6

Bias=2

2	1	1	3	3	4	5	5	6	9
---	---	---	---	---	---	---	---	---	---

1	1	2	3	3	4	5	5	6	9
---	---	---	---	---	---	---	---	---	---

Choose a bias (a random value between array min and max). Here 3

Sort in two arrays: less to bias (left) and greater or equal to bias (right)

Restart the same operation on the two subarrays.

Etc...

Advantage :
No insertion
Only swap

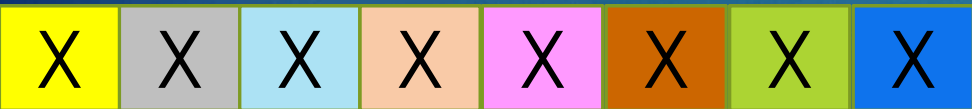
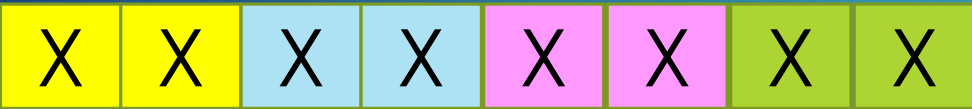
ACP

N comparisons



N/2 comp.

N/2 comp.



Qsort algorithm

127

Advantages of Qsort:

Complexity:
 $O(N \times \log_2(N))$

Use only swap (no insertion): array can be used (faster than chained list)

Can be parallelized

$\log_2(N)$ stages

- Generate an array of 1000000 random ints
- Sort the array by using YOUR qsort algorithm (not the one included in C)
- Write a function which verifies the array is correctly sorted

```
void swap(int array[], int a, int b) {
    int temp = array[a];
    array[a] = array[b];
    array[b] = temp;
}

void quickSort(int array[], int start, int end) {
    int left = start-1;
    int right = end+1;
    const int bias = array[start];
    /* if array length is null, nothing to do */
    if(start >= end) return;
    /* else, we read the array, one time from right
    to left and one time from left to right to search
    for element which are on the wrong place.
    Once found, we swap them. When right index
    reaches left index we stop.*/
```

```
    while(1) {
        do right--; while(array[right] > bias);
        do left++; while(array[left] < bias);
        if(left < right) swap(array, left, right);
        else break;
    }
    /* Now, all elements lesser than bias are on the
    left side and all elements greater than bias are
    on the right side. Thus, we have 2 groups to
    sort. We launch quicksort on these 2 groups!
    That's what we call recursivity ! */
    quickSort(array, start, right);
    quickSort(array, right+1, end);
}
```

ACP

Qsort (ex.)

130

```
void echanger(int tableau[], int a, int b) {  
    int temp = tableau[a];  
    tableau[a] = tableau[b];  
    tableau[b] = temp;  
}
```

```
void quickSort(int tableau[], int debut, int fin) {  
    int gauche = debut-1;  
    int droite = fin+1;  
    const int pivot = tableau[debut];
```

```
    /* Si le tableau est de longueur nulle, il n'y a rien à  
    faire. */  
    if(debut >= fin) return;
```

```
    /* Sinon, on parcourt le tableau, une fois de droite  
    à gauche, et une autre de gauche à droite, à la  
    recherche d'éléments mal placés, que l'on permute.  
    Si les deux parcours se croisent, on arrête. */
```

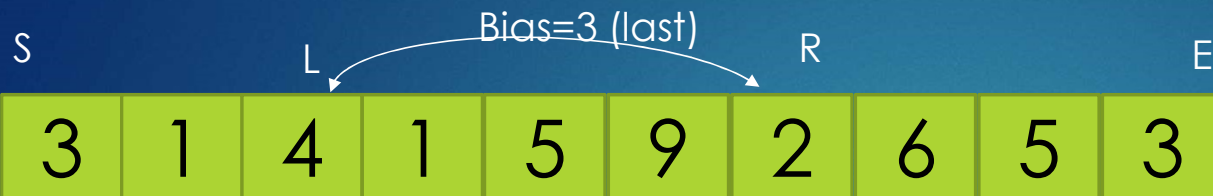
```
while(1) {  
    do droite--; while(tableau[droite] > pivot);  
    do gauche++; while(tableau[gauche] < pivot);  
    if(gauche < droite)  
        echanger(tableau, gauche, droite);  
    else break;  
}
```

```
/* Maintenant, tous les éléments inférieurs au  
pivot sont avant ceux supérieurs au pivot. On a  
donc deux groupes de cases à trier. On utilise  
pour cela... la méthode quickSort elle-même ! */  
quickSort(tableau, debut, droite);  
quickSort(tableau, droite+1, fin);  
}
```


ACP

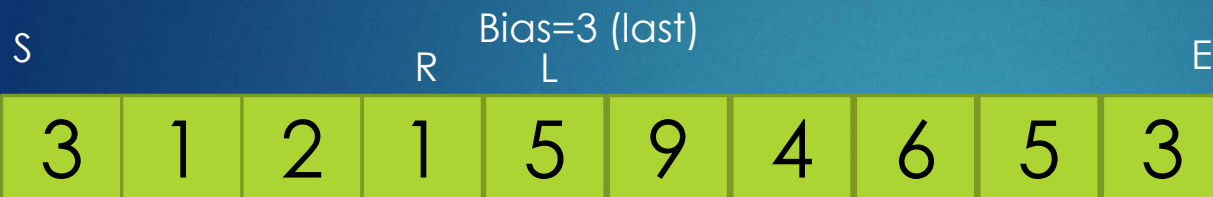
Qsort algorithm

131

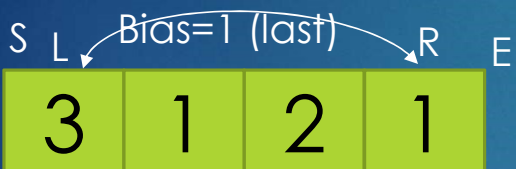


S=start E=end
L=Left R=Right

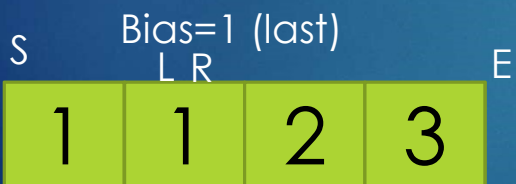
$R > L$
 $*R < \text{bias} * L > \text{bias}$
Xchange(*L, *R)



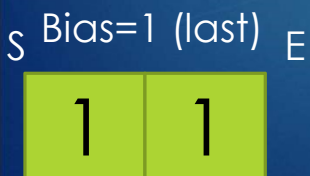
$R < L$
Qsort(S,R); Qsort(R+1,E)



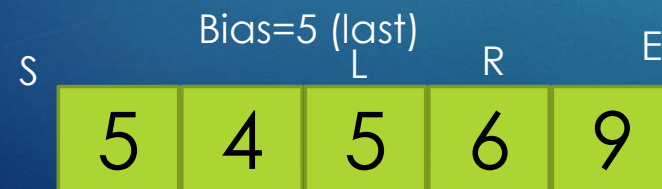
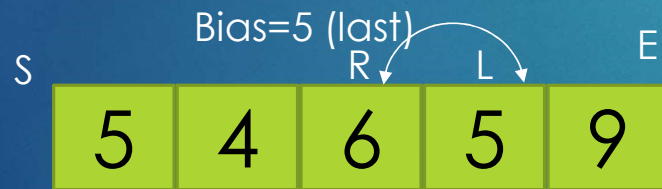
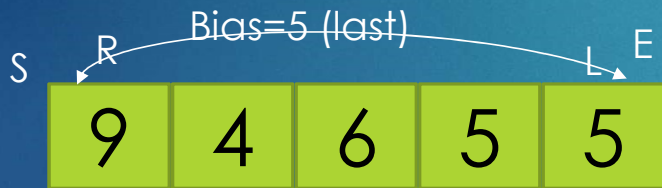
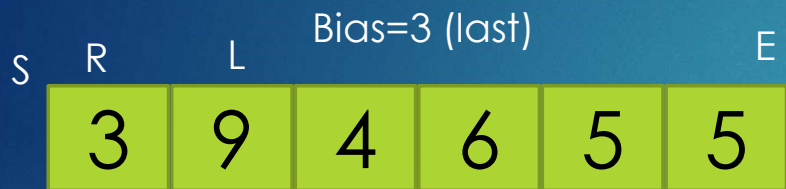
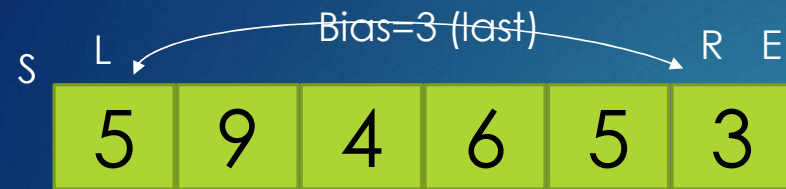
$R > L$
 $*R < \text{bias} * L > \text{bias}$
Xchange(*L, *R)



$R == L$
Qsort(S,R); Qsort(R+1,E)



ACP



Qsort algorithm

132

S=start E=end
L=Left R=Right

R>L
*R<bias *L>bias
Xchange(*L, *R)

R<L
Qsort(S,R); Qsort(R+1,E)

R>L
*R<bias *L>bias
Xchange(*L, *R)

R>L
*R<bias *L>bias
Xchange(*L, *R)

R<L
Qsort(S,R); Qsort(R+1,E)

ACP

Qsort (ex.)

133

Hint: to generate random numbers:

PseudoRandom:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main(void){
    int i = 0;
    int random_number = 0;
    for(i=0; i<5; i++) {
        random_number = rand();
        printf("%d ",random_number);
    }
    return 0;
}
```

Random:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main(void) {
    int i = 0;
    int random_number = 0;
    srand(time(NULL));
    for(i=0; i<5; i++){
        random_number = rand();
        printf("%d ",random_number);
    }
    return 0;
}
```

- register** May be applied to local variables.
It specify to the compiler to use the variable in a processor register .
- static** May be applied to local variables.
It specify to the compiler to store the variable in heap instead stack.
- volatile** May be applied to all variables.
It specify to the compiler to avoid code optimization on the variable. The variable may change without code instruction (hardware, thread, etc...)

Process

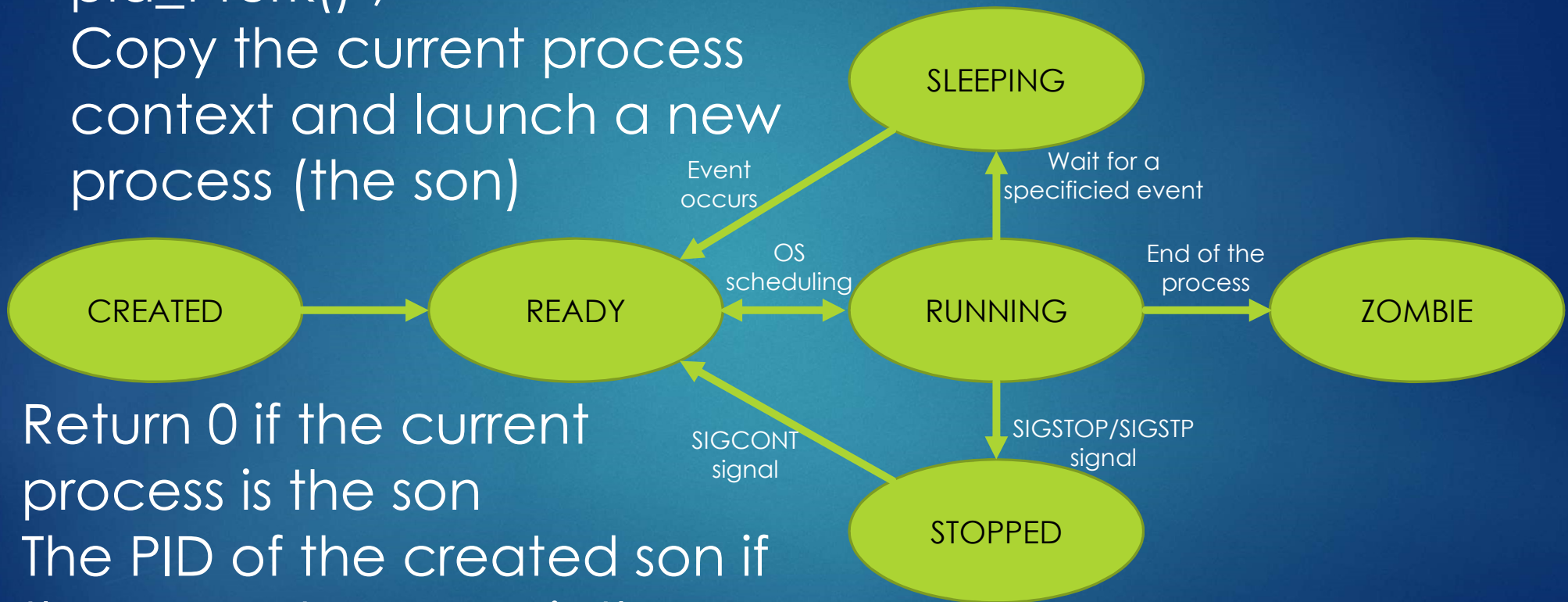
(fork)

ACP

```
#include <unistd.h>
```

```
pid_t fork() ;
```

Copy the current process context and launch a new process (the son)

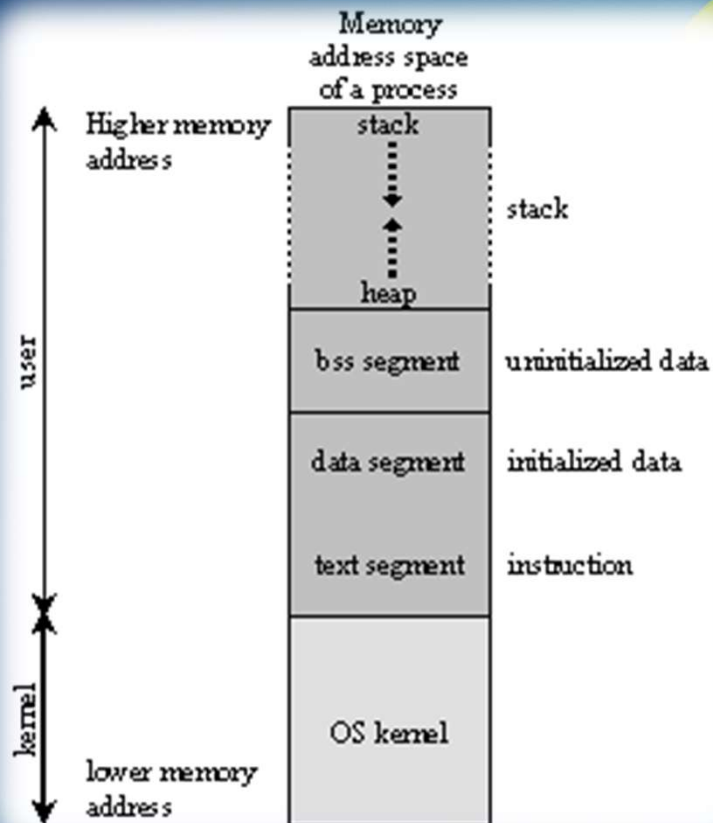


Return 0 if the current process is the son

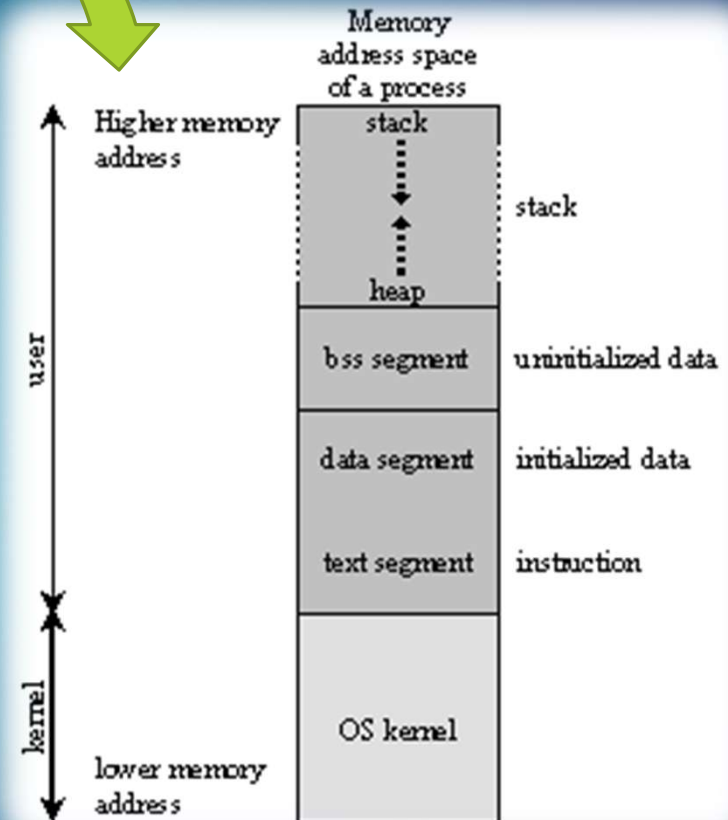
The PID of the created son if the current process is the parent

Process

136

Fork
Memory copy

Parent (father)



Child (son)

ACP

Getpid : get the current process ID

Getppid : get the parent (father) process ID

Process

138

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
int main (void) {
    pid_t pid ;
    printf(« Before the fork ;I'm the process %d my
    father is %d\n\n",getpid(), getppid()) ;
    pid = fork() ;
    printf("fork result %d\n",pid) ;
    printf("After the fork ; I'm the process %d my
    father is %d\n\n",getpid(), getppid()) ;
    return 0 ;
}
```

Output:

Before the fork; I'm the
process 11383 my father is
1258

Fork result 11384

After the fork; I'm the process
11383 my father is 1258

Fork result 0

After the fork; I'm the process
11384 my father is 11383

ACP

Pipes

139

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <stdlib.h>
#include <string.h>
int main(void){
int  fd[2], nbytes, windex;
pid_t  childpid;
char  string[] = "Luke I'm your father!\n";
char  string2[] = "Damned\n";
char  readbuffer[80];
pipe(fd);
if((childpid = fork()) == -1) {
    perror("fork"); exit(1);
}
if(childpid == 0) {
    // Luke is born
    // wait for full string
    windex=-1;
    do {
```

```
        windex++;
        // read is blocking it will be released when the
        character will be received
        nbytes = read(fd[0], &readbuffer[windex], 1);
    } while(readbuffer[windex] != 0);
    printf("Son Received string: %s\n", readbuffer);
    write(fd[1], string2, (strlen(string2)+1));
} else {
    /* Send "string" through the output side of pipe */
    write(fd[1], string, (strlen(string)+1));
    sleep(1); // avoid reading before son
    // wait for full string
    windex=-1;
    do {
        windex++; // read is blocking it will be released
        when the character will be received
        nbytes = read(fd[0], &readbuffer[windex], 1);
    } while(readbuffer[windex] != 0);
    printf("Father: Received string: %s\n", readbuffer);
}
}
```

Pthreads

A thread is a process (stream of instructions)

A program can create several threads

Threads share the same memory heap

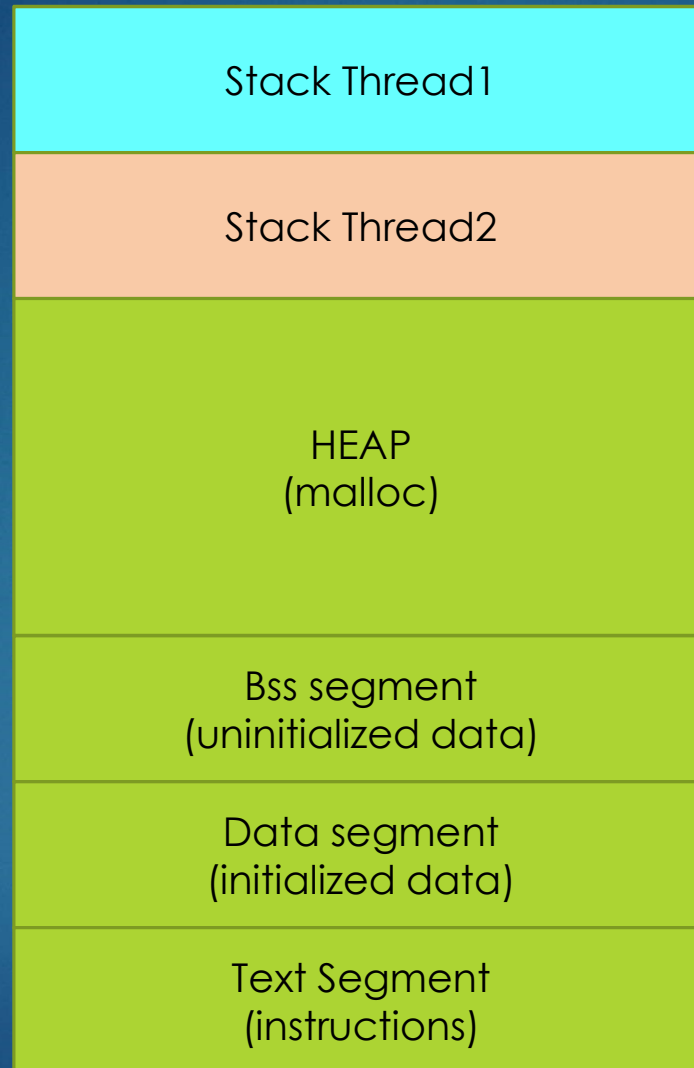
Thread are scheduled by the OS
(preemptive threads)

Very light
compared to
processes

ACP

Pthread

142




```
#include <pthread.h>
```

```
int pthread_create (pthread_t *thread, pthread_attr_t *attr, void *(*start_routine) (void *), void *arg);
```

```
pthread_t pthread1;
```

```
Int main() {  
    int ret, i=2;  
    ret=pthread_create(&pthread1, NULL, pthread1Func, (void *)i);  
    if(ret!=0) {  
        printf (stderr, "%s", strerror (ret));  
    }  
}
```

```
Void *pthread1Func(void *par) {  
    int i=(int)par;  
    ...  
    return NULL;  
}
```

```
#include <pthread.h>
```

```
int pthread_join (pthread_t th, void **thread_return);
```

```
pthread_t pthread1;
```

```
Int main() {  
    int ret;  
    ret=pthread_create(&pthread1, NULL, pthread1Func, NULL);  
    if(ret!=0) {  
        printf (stderr, "%s", strerror (ret));  
    }  
    pthread_join(pthread1, NULL);  
}
```

```
Void *pthread1Func(void *) {  
    ...  
    return NULL;  
}
```

The calling thread sleeps until the end of the specified thread.

Returns :

0 if OK

ESRCH if no thread match with the specified thread

EINVAL if another thread is already waiting for the end of **th**

EDEADLK if **th** is the current thread

ACP

Pthread (ex.)

145

- Write a program which creates ten threads (stored in a pthread array or a chain list).
- Each thread will loop n times. “n” is an integer which will be given as a parameter to the created pthread (using the parameter of pthread_create).
- The main program will wait the end of all threads in order to finish.
- You will use printf in order to display when the threads start and when the threads end.

ACP

Pthread (ex.)

146

- Write a program which creates ten threads (stored in a pthread array or a chain list).
- Each thread will increment 1000000 times a shared global variable.
- The main program will wait the end of all threads in order to finish and will print the shared global variable.
- Did you expect this result?

ACP

Pthread

147

```
int pthread_mutex_lock (pthread_mutex_t * mutex);
```

Lock a mutex (mutual exclusion)

If the mutex is free (=0): the current thread takes the mutex (=1).

If the current thread is already the owner of the mutex is incremented (+1).

If the mutex is already used by another thread, the current thread is set in a sleeping state. It takes no more CPU (passive wait). The thread will be awakened when the mutex will be freed.

return 0 if OK


```
int pthread_mutex_unlock (pthread_mutex_t * mutex);
```

Unlock a mutex

Decrement the mutex counter (-1).

If the counter is equal 0 the mutex is freed and the next thread waiting (sleeping) for this mutex is awoken.

The algorithm may be system (OS) dependent.

return 0 if OK

ACP

Pthread

149

```
pthread_mutex_t
```

```
pmutex_var1=PTHREAD_MUTEX_INITIALIZER;
```

```
Int var1
```

```
void thread1Func(void *) {  
    while(1){  
        pthread_mutex_lock(&pmutex_var1  
);  
        var1++;  
        pthread_mutex_unlock(&pmutex_v  
ar1);  
    }  
}
```

```
void thread2Func(void *) {  
    while(1){  
        pthread_mutex_lock(&pmutex_var1  
);  
        var1++;  
        pthread_mutex_unlock(&pmutex_v  
ar1);  
    }  
}
```

- Take your Qsort code
- How much time does it take to sort an array of 100000 elements?
- Multithread qsort : Parallelize your algorithm by creating a thread each time you launch *qsort*.
- (Hint) try to use only local variables to avoid globals (and mutex).
- Compare execution time with the monothread *qsort* (on several machines).


```
void echanger(int tableau[], int a, int b) {  
    int temp = tableau[a];  
    tableau[a] = tableau[b];  
    tableau[b] = temp;  
}
```

```
void quickSort(int tableau[], int debut, int fin) {  
    int gauche = debut-1;  
    int droite = fin+1;  
    const int pivot = tableau[debut];
```

```
    /* Si le tableau est de longueur nulle, il n'y a rien à  
    faire. */  
    if(debut >= fin) return;
```

```
    /* Sinon, on parcourt le tableau, une fois de droite  
    à gauche, et une autre de gauche à droite, à la  
    recherche d'éléments mal placés, que l'on permute.  
    Si les deux parcours se croisent, on arrête. */
```

```
while(1) {  
    do droite--; while(tableau[droite] > pivot);  
    do gauche++; while(tableau[gauche] < pivot);  
    if(gauche < droite)  
        echanger(tableau, gauche, droite);  
    else break;  
}
```

```
    /* Maintenant, tous les éléments inférieurs au pivot sont  
    avant ceux supérieurs au pivot. On a donc deux groupes  
    de cases à trier. On utilise  
    pour cela... la méthode quickSort elle-même ! */  
    quickSort(tableau, debut, droite); // Create Pthread1  
    quickSort(tableau, droite+1, fin); // Create Pthread2  
    // wait for end of Pthread1 & Pthread2 here  
}
```

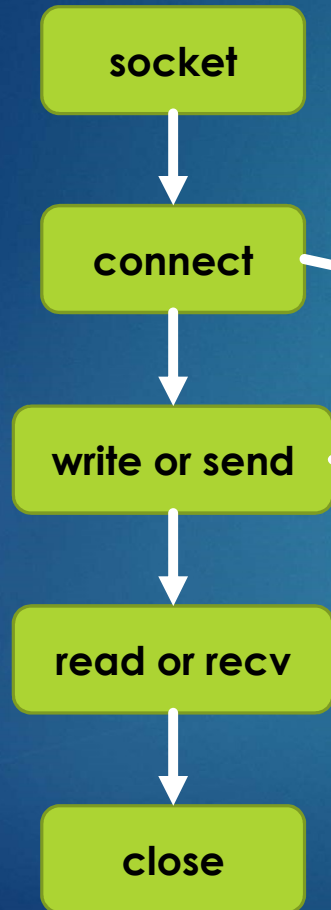

Sockets

ACP

TCP Sockets

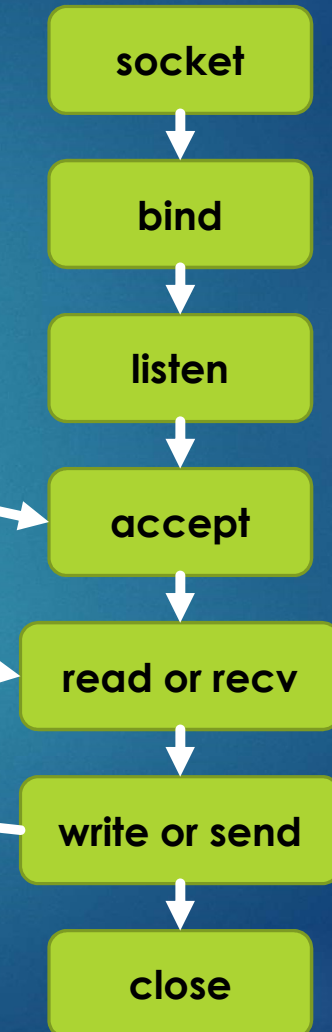
153

Client



Ethernet

Server



```
/*
 * C ECHO client example using sockets
 */
#include<stdio.h> //printf
#include<string.h> //strlen
#include<sys/socket.h> //socket
#include<arpa/inet.h> //inet_addr

int main(int argc , char *argv[]) {
    int sock;
    struct sockaddr_in server;
    char message[1000] , server_reply[2000];

    //Create socket
    sock = socket(AF_INET , SOCK_STREAM , 0);
    if (sock == -1) { printf("Could not create socket"); }
    puts("Socket created");

    // Define the socket remote address : 127.0.0.1:8888
    server.sin_addr.s_addr = inet_addr("127.0.0.1");
    server.sin_family = AF_INET;
    server.sin_port = htons( 8888 );

    //Connect to remote server
    if (connect(sock , (struct sockaddr *)&server , sizeof(server)) < 0) {
        perror("connect failed. Error");
        return 1;
    }
    puts("Connected\n");
```

```
//keep communicating with server
while(1) {
    printf("Enter message : ");
    scanf("%s" , message);

    //Send some data
    if( write(sock , message , strlen(message) , 0) < 0) {
        puts("write failed");
        return 1;
    }

    //Receive a reply from the server
    if( read(sock , server_reply , 2000) < 0) {
        puts("read failed");
        break;
    }

    puts("Server reply :");
    puts(server_reply);
}

close(sock);
return 0;
}
```

```
Sock=socket(AF_INET, SOCK_STREAM, 0);
```

Arg1:

AF_INET : Ethernet Socket

AF_UNIX : Unix Socket

Arg2:

SOCK_STREAM : TCP Socket (safe/slow)

SOCK_DGRAM : UDP Socket (unsafe/fast)

```
server.sin_addr.s_addr = inet_addr("127.0.0.1");
```

"127.0.0.1" : local address (loopback (lo))

"www.google.com"

```
server.sin_port = htons( 8888 );
```

8888 : port number. 2 bytes : 0 – 16535

80 : html port

```
write(sock , message , strlen(message) )
```

sock : socket used to write data

message: buffer containing data to send

strlen(message) : number of bytes to send

```
read(sock , server_reply , 10)
```

sock : socket used to read data

server_reply: buffer used to put data coming from socket

10 : number of bytes to read from the socket



Read will block until all bytes have been read

ACP

```
/* C socket server example */
#include<stdio.h>
#include<string.h> //strlen
#include<sys/socket.h>
#include<arpa/inet.h> //inet_addr
#include<unistd.h> //write

int main(int argc , char *argv[]) {
    int socket_desc , client_sock , c , read_size;
    struct sockaddr_in server , client;
    char client_message[2000];

    //Create socket
    socket_desc = socket(AF_INET , SOCK_STREAM , 0);
    if (socket_desc == -1) {
        printf("Could not create socket");
    }
    puts("Socket created");

    //Prepare the sockaddr_in structure
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons( 8888 );

    //Bind
    if (bind(socket_desc,(struct sockaddr *)&server , sizeof(server)) < 0) {
        //print the error message
        perror("bind failed. Error");
        return 1;
    }
    puts("bind done");
```

TCP Server

156

```
//Listen
listen(socket_desc , 3);

//Accept and incoming connection
puts("Waiting for incoming connections...");
c = sizeof(struct sockaddr_in);

//accept connection from an incoming client
client_sock = accept(socket_desc, (struct sockaddr *)&client,
(socklen_t *)&c);
if (client_sock < 0) {
    perror("accept failed");
    return 1;
}
puts("Connection accepted");

//Receive a message from client
while( (read_size = read(client_sock , client_message , 2000)) > 0 ) {
    //Send the message back to client
    write(client_sock , client_message , strlen(client_message));
}

if(read_size == 0) {
    puts("Client disconnected");
    fflush(stdout);
} else if(read_size == -1) {
    perror("recv failed");
}
return 0;
}
```

```
server.sin_family = AF_INET;  
server.sin_addr.s_addr = INADDR_ANY;  
server.sin_port = htons( 8888 );  
bind(socket_desc,(struct sockaddr *)&server , sizeof(server))
```

⇒ Set the socket parameter and affect them to the socket descriptor

```
listen(socket_desc , 3);
```

⇒ Wait for incoming sockets (clients). 3 sockets can be opened simultaneously



Listen will block until a client incoming connection occurs

```
client_sock = accept(socket_desc, (struct sockaddr *)&client, (socklen_t*)&c);
```

⇒ Accept the incoming connection and affect it to the new descriptor « client_sock ». « client_sock » will be used to communicate with the new client. « server » can be used to wait another incoming connection.