C Programming
Day09.B

2017.10.17

string
array
recursion

```
Char *s  = "Hello, world!" ;
Char s[16] = "Hello, world!" ;
int  a  =  30
```

↑ variable or consecutive variables

↑ constant

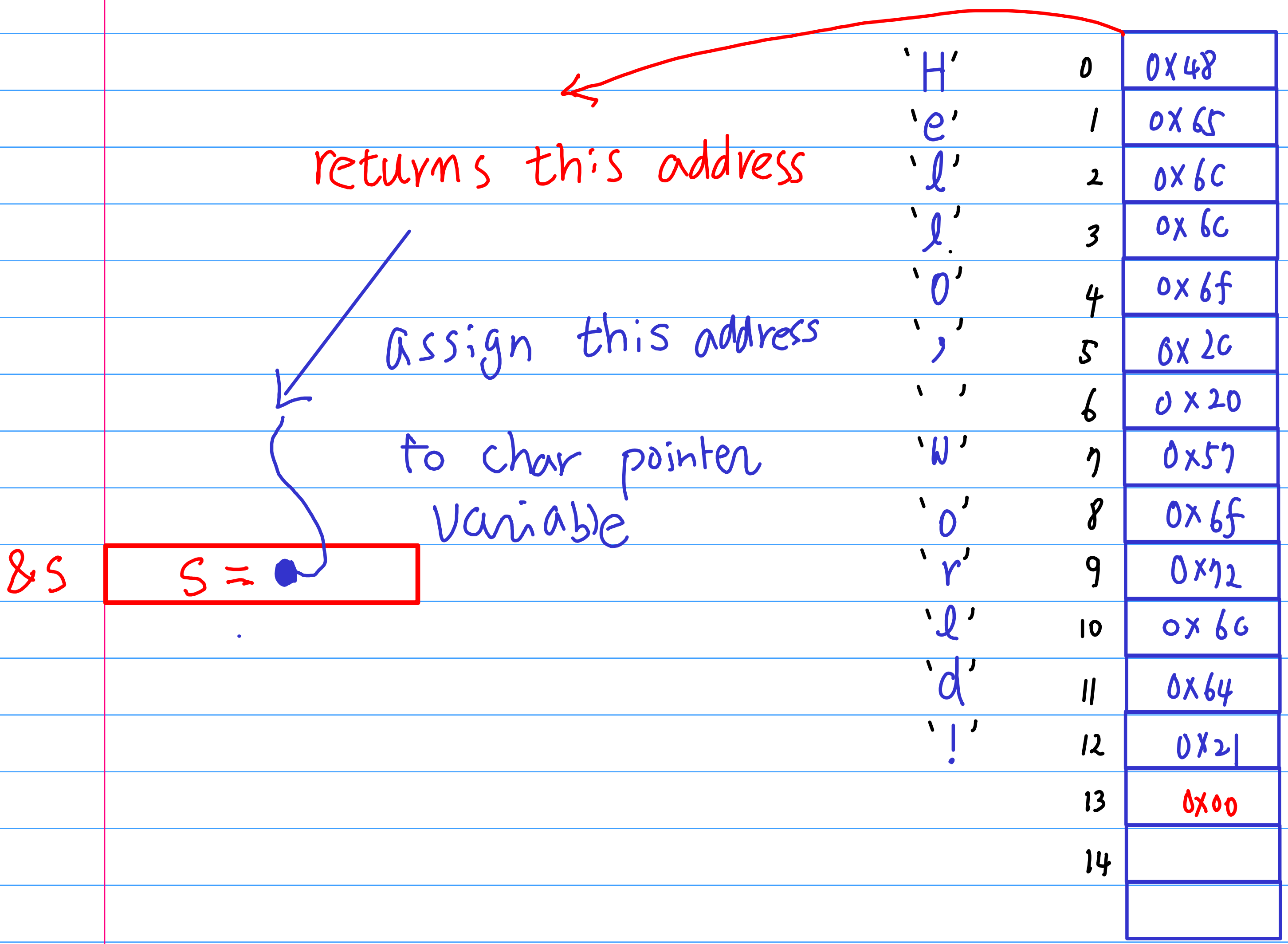
☆ 30 integer constant

30 = 16 + 14 ⇒ 0X1E

integer 4 bytes ⇒ 0X 00 00 00 1E

00
00
00
1E

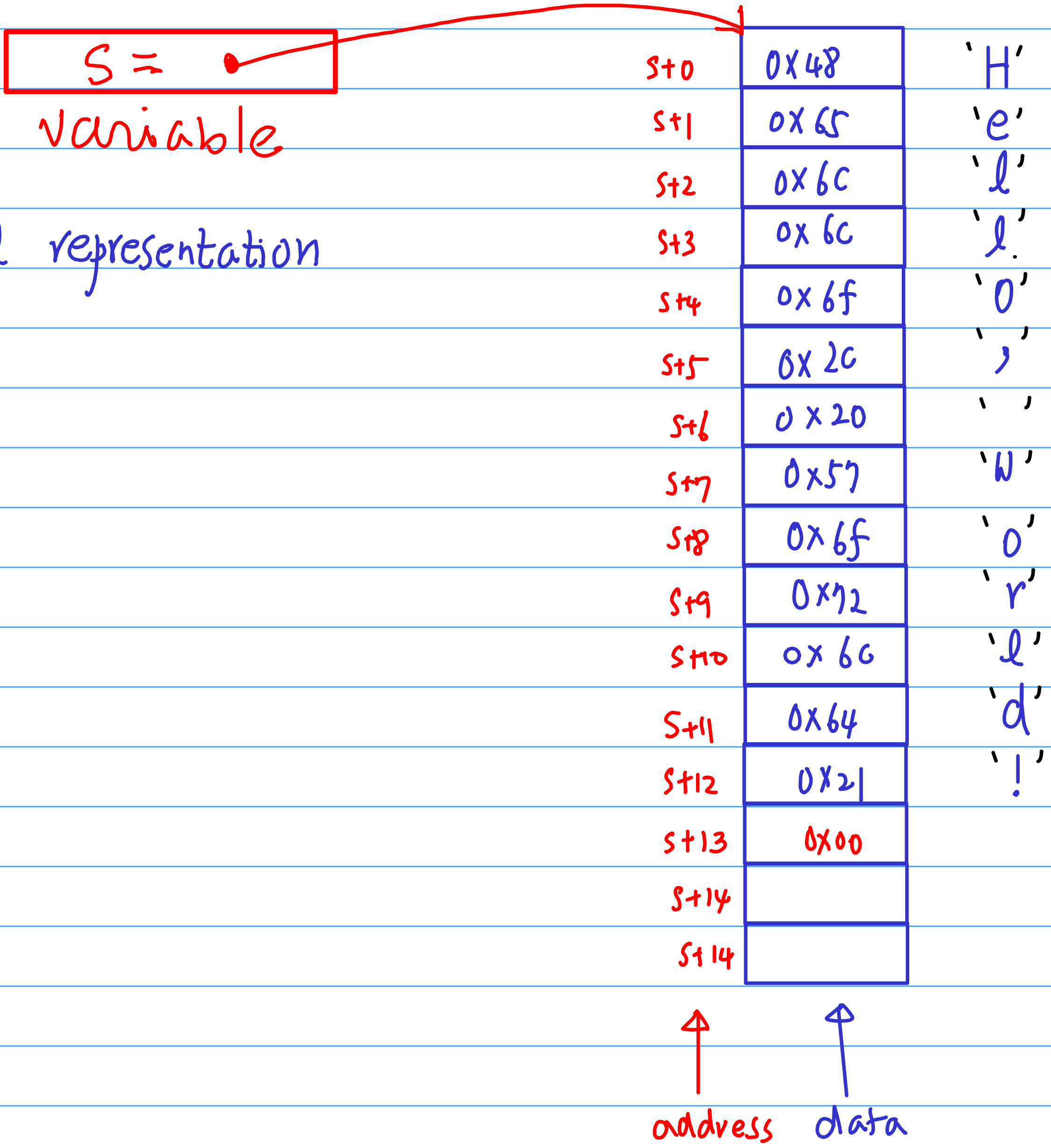
☆ "Hello, world!" string constant



'H'	0	0x48
'e'	1	0x65
'l'	2	0x6c
'l'	3	0x6c
'o'	4	0x6f
','	5	0x2c
' '	6	0x20
'W'	7	0x57
'o'	8	0x6f
'r'	9	0x72
'l'	10	0x6c
'd'	11	0x64
'!'	12	0x21
	13	0x00
	14	

"Hello, world!" string constant

Usual representation



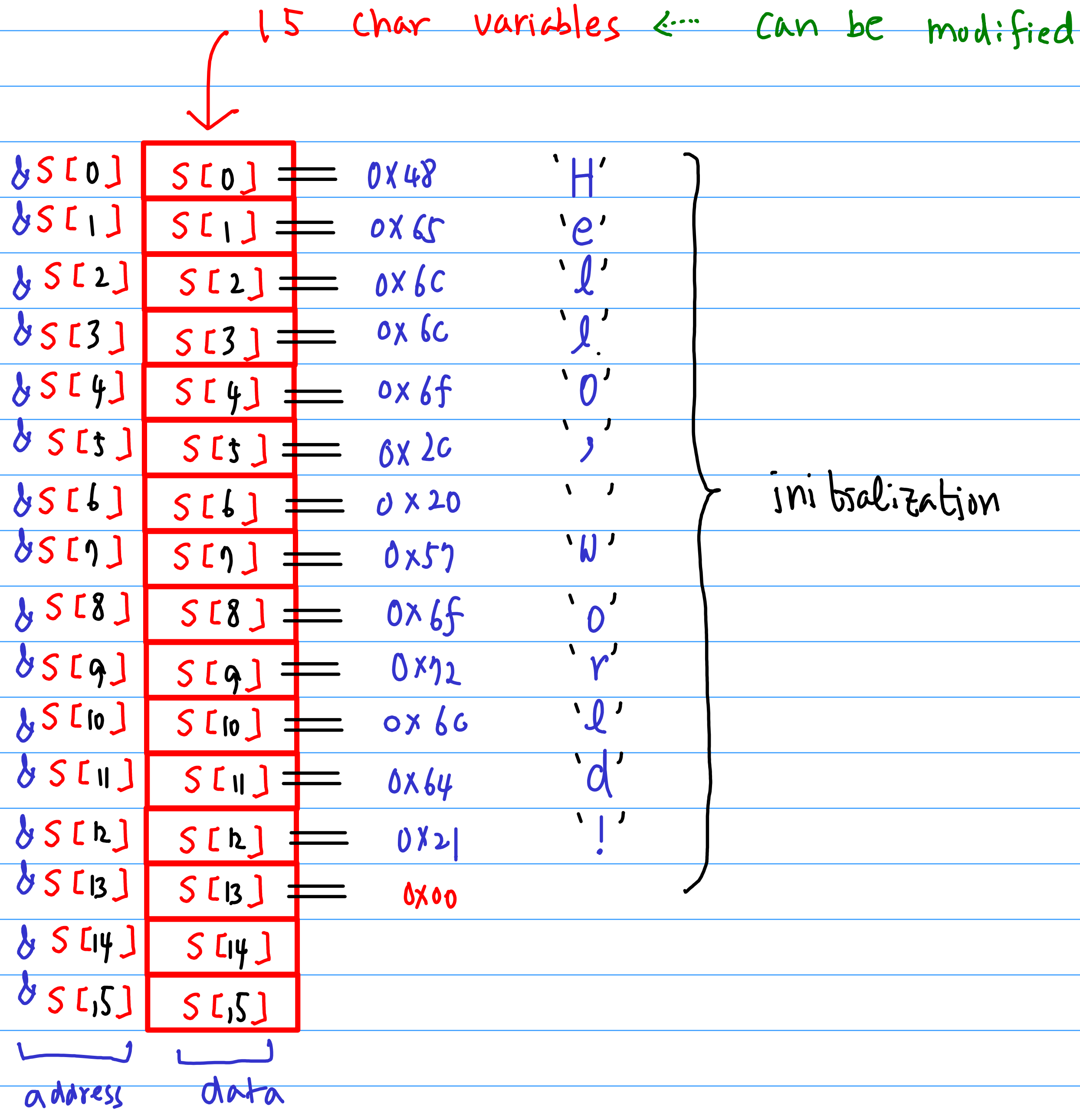
s+0	0x48	'H'
s+1	0x65	'e'
s+2	0x6c	'l'
s+3	0x 6c	'l'
s+4	0x 6f	'o'
s+5	0x 2c	','
s+6	0x20	' '
s+7	0x57	'W'
s+8	0x 6f	'o'
s+9	0x72	'r'
s+10	0x 6c	'l'
s+11	0x64	'd'
s+12	0x21	'!'
s+13	0x00	
s+14		
s+14		

can't change

∵ string constant

char *s = "Hello, world!" ;

```c
char s[16] = "Hello, world!" ;
```



15 char variables ⟵ can be modified

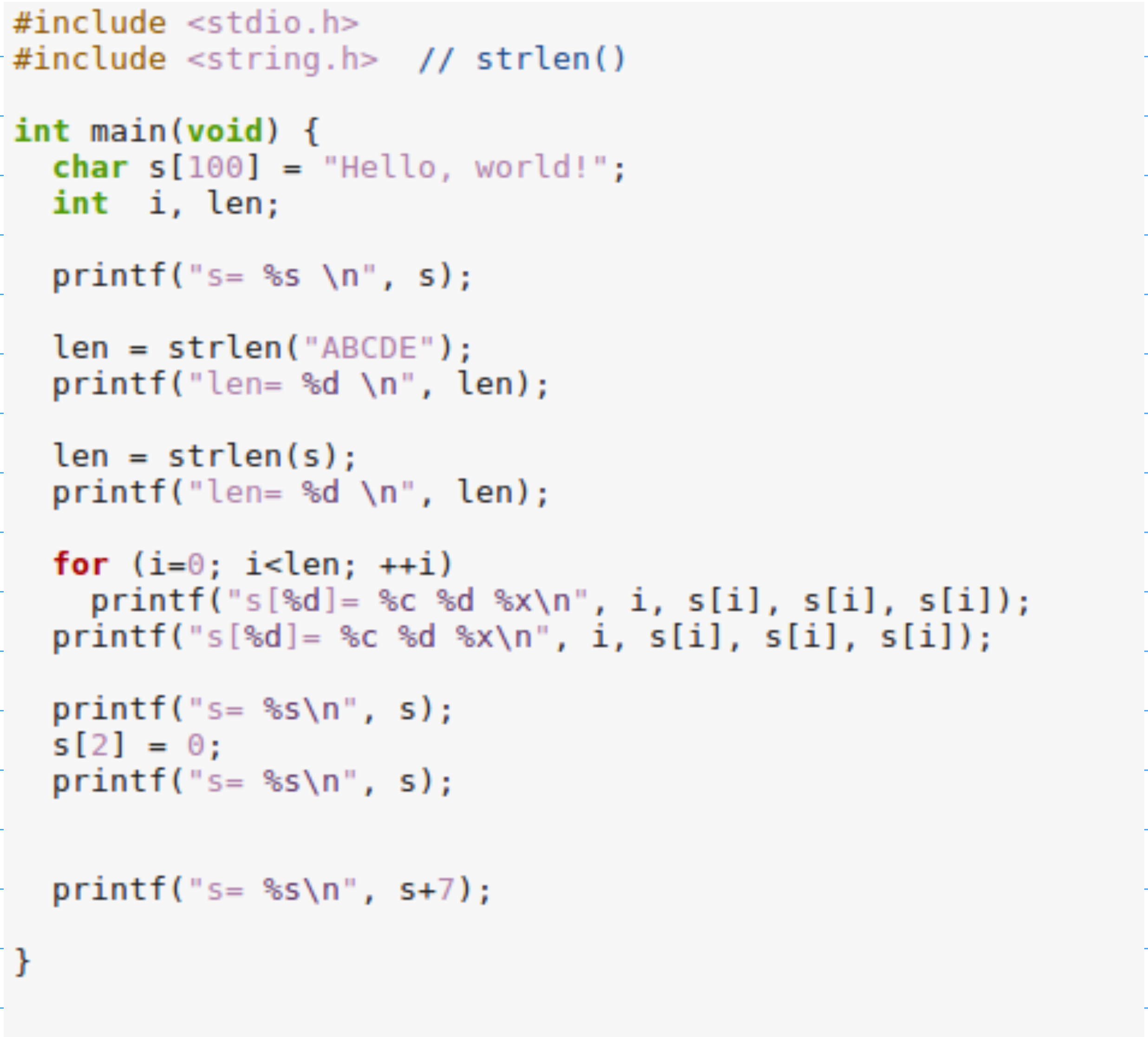
address	data		
&s[0]	s[0]	0x48	'H'
&s[1]	s[1]	0x65	'e'
&s[2]	s[2]	0x6c	'l'
&s[3]	s[3]	0x 6c	'l'
&s[4]	s[4]	0x 6f	'o'
&s[5]	s[5]	0x 2c	','
&s[6]	s[6]	0x20	' '
&s[7]	s[7]	0x57	'W'
&s[8]	s[8]	0x 6f	'o'
&s[9]	s[9]	0x72	'r'
&s[10]	s[10]	0x 6c	'l'
&s[11]	s[11]	0x64	'd'
&s[12]	s[12]	0x21	'!'
&s[13]	s[13]	0x00	
&s[14]	s[14]		
&s[15]	s[15]		

initialization

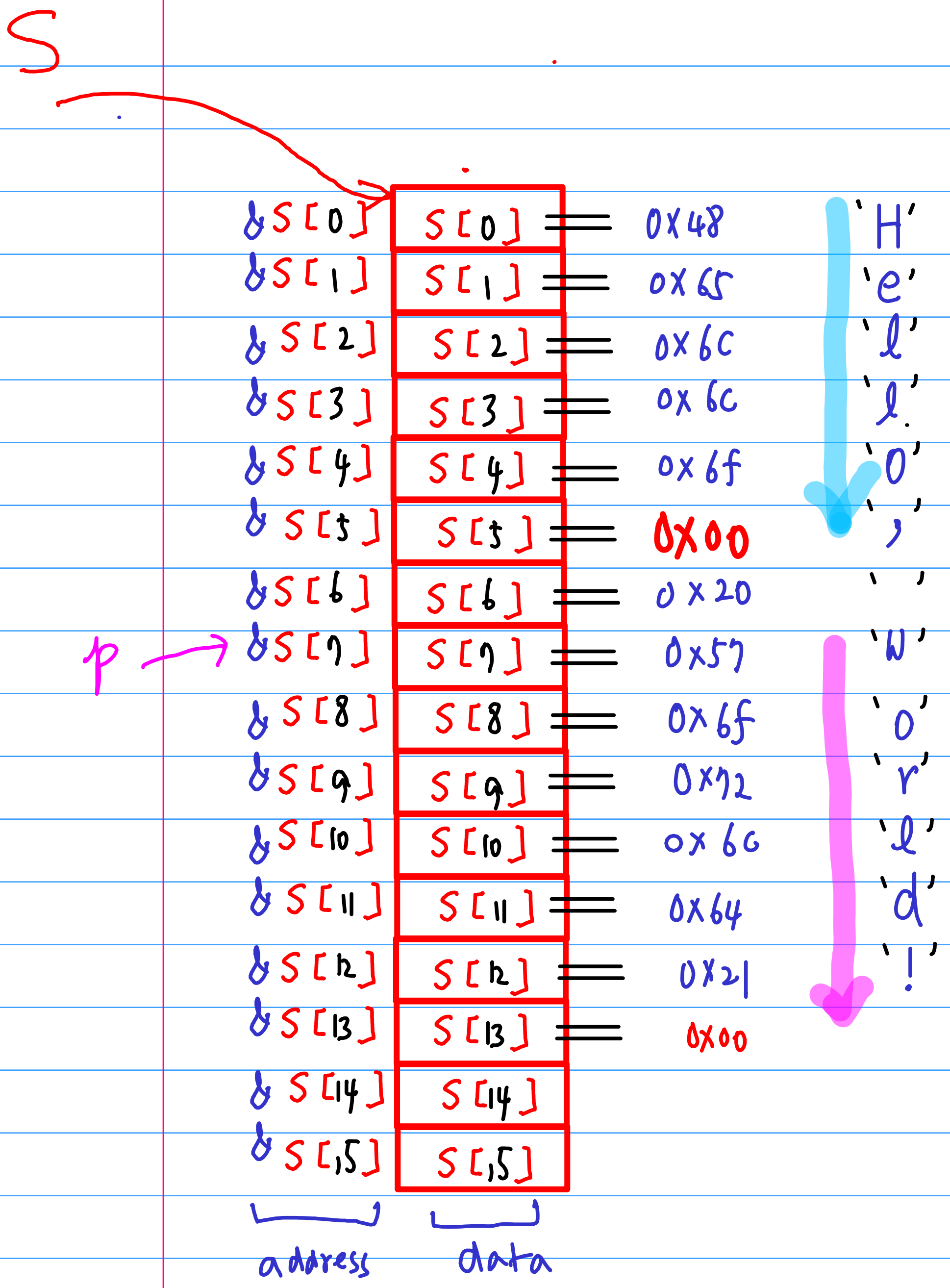
char s[16] = "Hello, world!" ;

15 char variables ⟵ can be modified

s+0	*(s+0)	==	0x48	'H'	initialization
s+1	*(s+1)	==	0x65	'e'	
s+2	*(s+2)	==	0x6c	'l'	
s+3	*(s+3)	==	0x6c	'l'	
s+4	*(s+4)	==	0x6f	'o'	
s+5	*(s+5)	==	0x2c	','	
s+6	*(s+6)	==	0x20	' '	
s+7	*(s+7)	==	0x57	'W'	
s+8	*(s+8)	==	0x6f	'o'	
s+9	*(s+9)	==	0x72	'r'	
s+10	*(s+10)	==	0x6c	'l'	
s+11	*(s+11)	==	0x64	'd'	
s+12	*(s+12)	==	0x21	'!'	
s+13	*(s+13)	==	0x00		
s+14	*(s+14)				
s+15	*(s+15)				
address	data				



```c
#include <stdio.h>
#include <string.h>  // strlen()

int main(void) {
  char s[100] = "Hello, world!";
  int  i, len;

  printf("s= %s \n", s);

  len = strlen("ABCDE");
  printf("len= %d \n", len);

  len = strlen(s);
  printf("len= %d \n", len);

  for (i=0; i<len; ++i)
    printf("s[%d]= %c %d %x\n", i, s[i], s[i], s[i]);
  printf("s[%d]= %c %d %x\n", i, s[i], s[i], s[i]);

  printf("s= %s\n", s);
  s[2] = 0;
  printf("s= %s\n", s);


  printf("s= %s\n", s+7);

}
```



&s[0]	s[0]	0x48	'H'
&s[1]	s[1]	0x65	'e'
&s[2]	s[2]	0x6c	'l'
&s[3]	s[3]	0x6c	'l'
&s[4]	s[4]	0x6f	'o'
&s[5]	s[5]	0x00	','
&s[6]	s[6]	0x20	' '
&s[7]	s[7]	0x57	'W'
&s[8]	s[8]	0x6f	'o'
&s[9]	s[9]	0x72	'r'
&s[10]	s[10]	0x6c	'l'
&s[11]	s[11]	0x64	'd'
&s[12]	s[12]	0x21	'!'
&s[13]	s[13]	0x00	
&s[14]	s[14]		
&s[15]	s[15]		

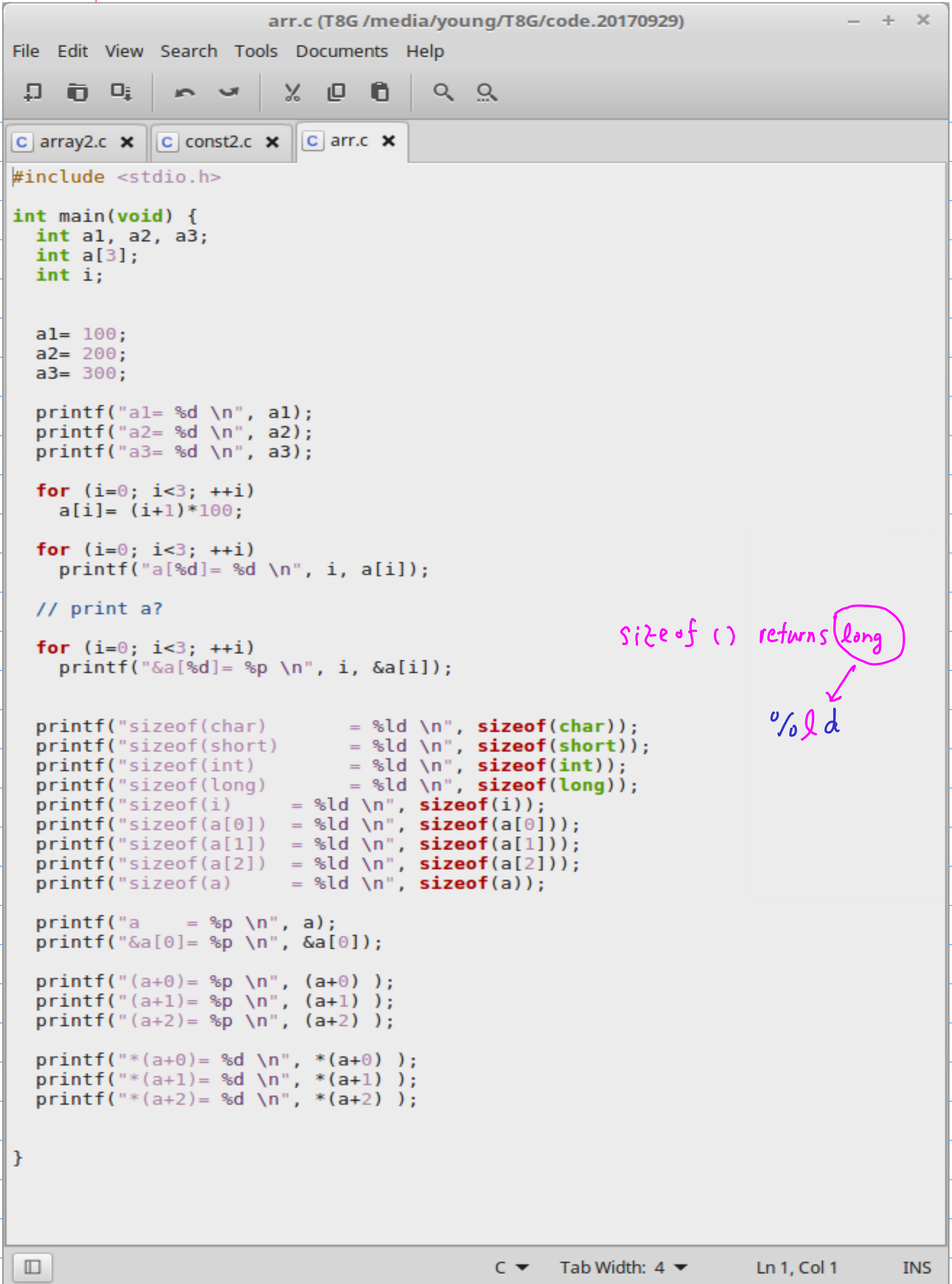
s[5] = 0

```
printf("%s \n", s);
```

→ Hello

```
char *p;
p = &s[7];

printf("%s \n", p );
```

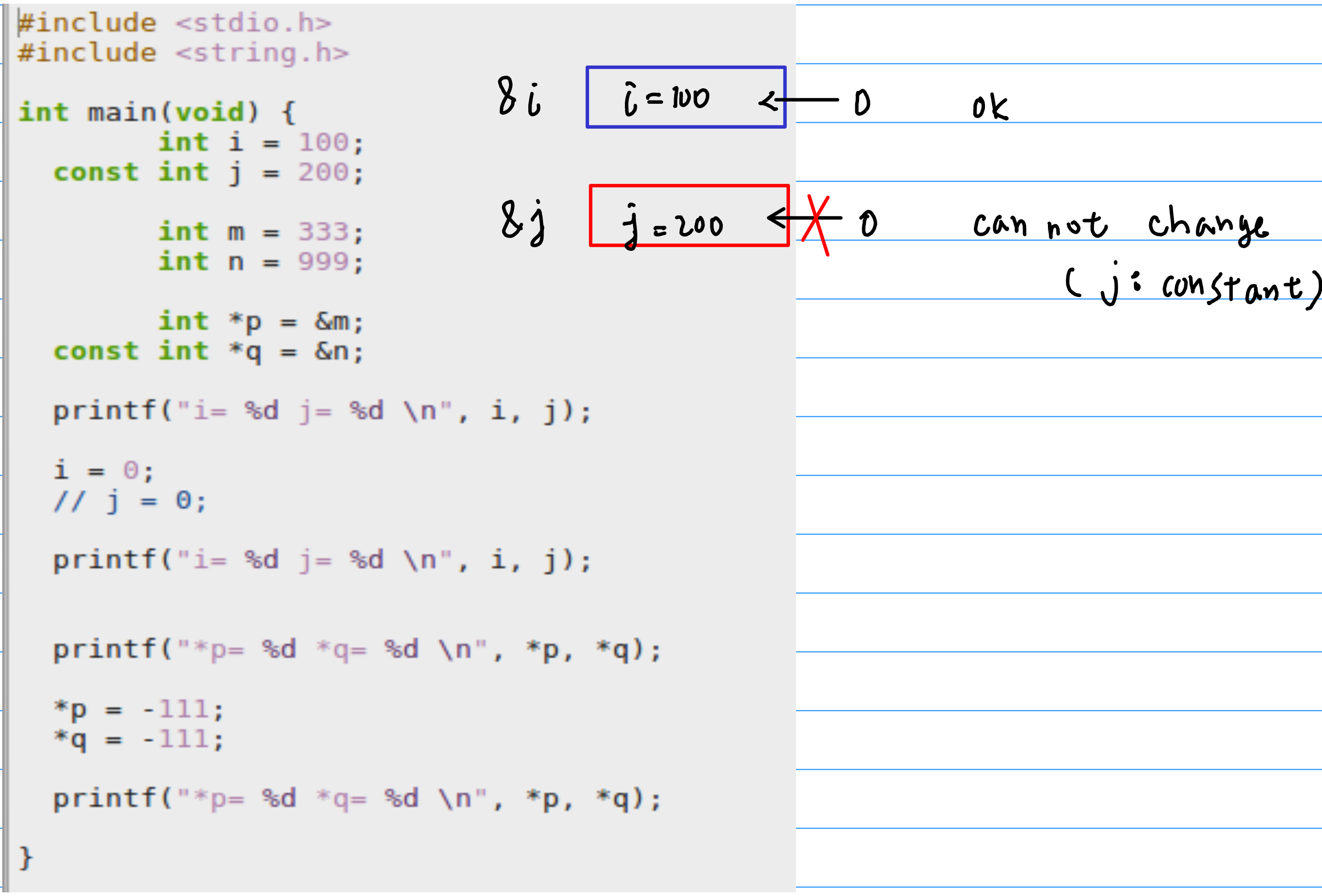
→ World !



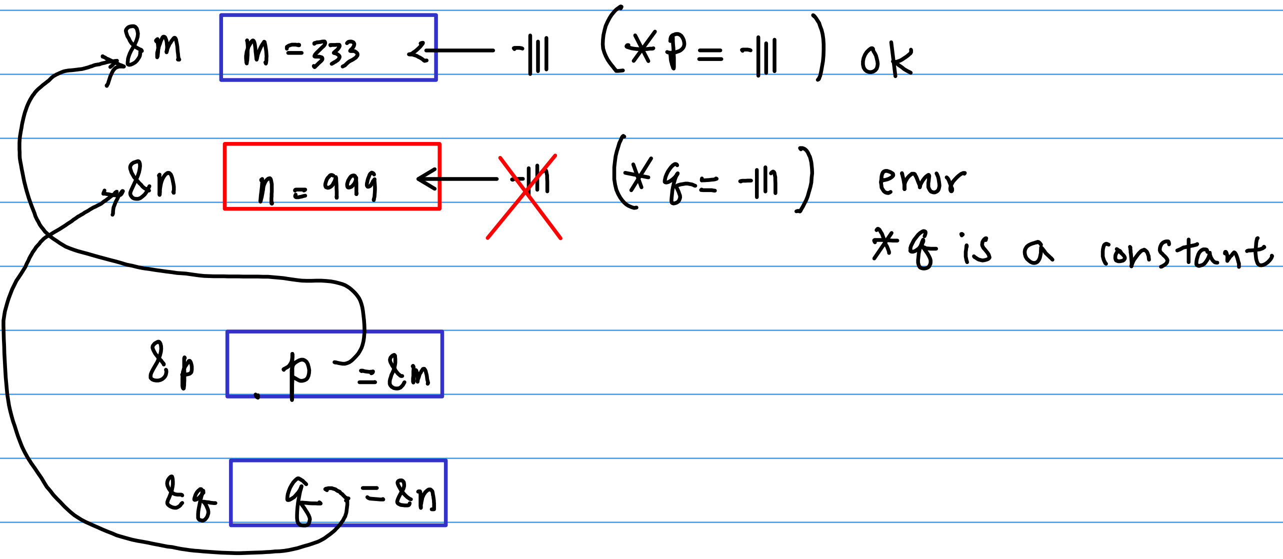
```
#include <stdio.h>

int main(void) {
  int a1, a2, a3;
  int a[3];
  int i;


  a1= 100;
  a2= 200;
  a3= 300;

  printf("a1= %d \n", a1);
  printf("a2= %d \n", a2);
  printf("a3= %d \n", a3);

  for (i=0; i<3; ++i)
    a[i]= (i+1)*100;

  for (i=0; i<3; ++i)
    printf("a[%d]= %d \n", i, a[i]);

  // print a?

  for (i=0; i<3; ++i)
    printf("&a[%d]= %p \n", i, &a[i]);


  printf("sizeof(char)       = %ld \n", sizeof(char));
  printf("sizeof(short)      = %ld \n", sizeof(short));
  printf("sizeof(int)        = %ld \n", sizeof(int));
  printf("sizeof(long)       = %ld \n", sizeof(long));
  printf("sizeof(i)     = %ld \n", sizeof(i));
  printf("sizeof(a[0])  = %ld \n", sizeof(a[0]));
  printf("sizeof(a[1])  = %ld \n", sizeof(a[1]));
  printf("sizeof(a[2])  = %ld \n", sizeof(a[2]));
  printf("sizeof(a)     = %ld \n", sizeof(a));

  printf("a    = %p \n", a);
  printf("&a[0]= %p \n", &a[0]);

  printf("(a+0)= %p \n", (a+0) );
  printf("(a+1)= %p \n", (a+1) );
  printf("(a+2)= %p \n", (a+2) );

  printf("*(a+0)= %d \n", *(a+0) );
  printf("*(a+1)= %d \n", *(a+1) );
  printf("*(a+2)= %d \n", *(a+2) );


}
```

sizeof () returns long

%ld



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

int main(void) {
        int i = 100;
  const int j = 200;

        int m = 333;
        int n = 999;

        int *p = &m;
  const int *q = &n;

  printf("i= %d j= %d \n", i, j);

  i = 0;
  // j = 0;

  printf("i= %d j= %d \n", i, j);


  printf("*p= %d *q= %d \n", *p, *q);

  *p = -111;
  *q = -111;

  printf("*p= %d *q= %d \n", *p, *q);
}
```



3 Types of Functions

```
int func1( int a) {

   a *= 999;
   return a;
}
```

```
int func2( int a) {

   if (a < 0) return -a;
   else         return a ;

}
```

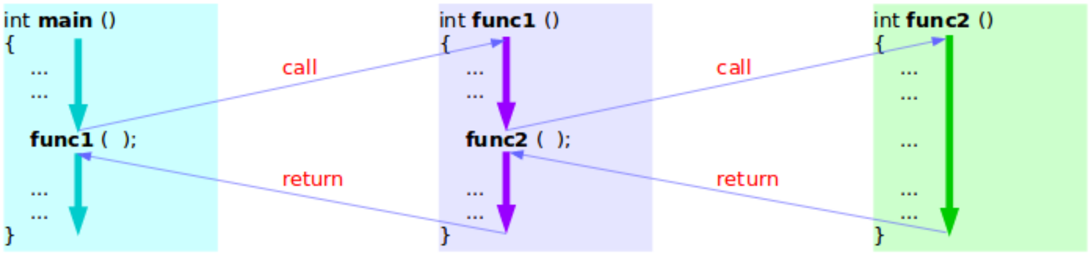
```
void func3( int a) {

   printf("%d \n", a) ;
   // return;
}
```

```
S= func1(100);
```

```
S= func2(100);
```

```
func3( 100 );
```

Indirect Function Call

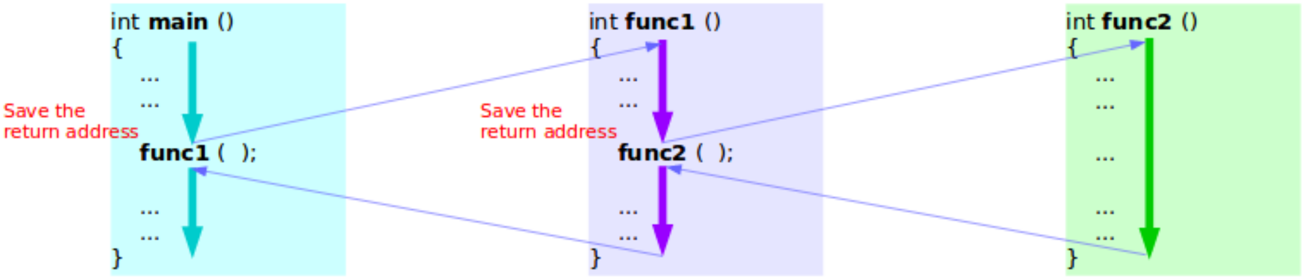


```
int main ()
{
    ...
    ...
    func1 ( );
    ...
    ...
}
```

```
int func1 ()
{
    ...
    ...
    func2 ( );
    ...
    ...
}
```

```
int func2 ()
{
    ...
    ...
    ...
    ...
    ...
}
```

Indirect Function Call



```
int main ()
{
   ...
   ...
   func1 ( );

   ...
   ...
}
```

Save the
return address

```
int func1 ()
{
   ...
   ...
   func2 ( );

   ...
   ...
}
```

Save the
return address

```
int func2 ()
{
   ...
   ...

   ...

   ...
   ...
}
```

main's local variables
on the stack frame

while main is active

Func1's local variables
on the stack frame

While func1 is active

Local Variables

Local Variables in a Stack Frame

```
int main (void)
{
        int S1 = 0;

➡       printf("S1 = %d \n", S);
        S1 = psum ( g );
        printf("S1 = %d \n", S);

        return 0;
}
```

```
int main (void)
{
        int S1 = 0;

        printf("S1 = %d \n", S);
➡       S1 = psum ( g );
        printf("S1 = %d \n", S);

        return 0;
}
```

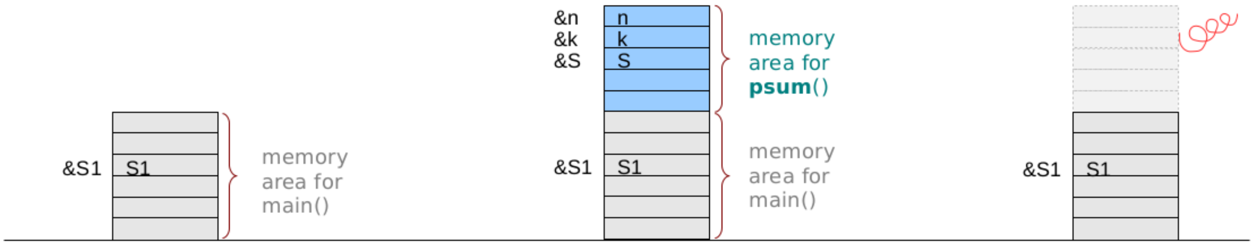
```
int main (void)
{
        int S1 = 0;

        printf("S1 = %d \n", S);
        S1 = psum ( g );
➡       printf("S1 = %d \n", S);

        return 0;
}
```

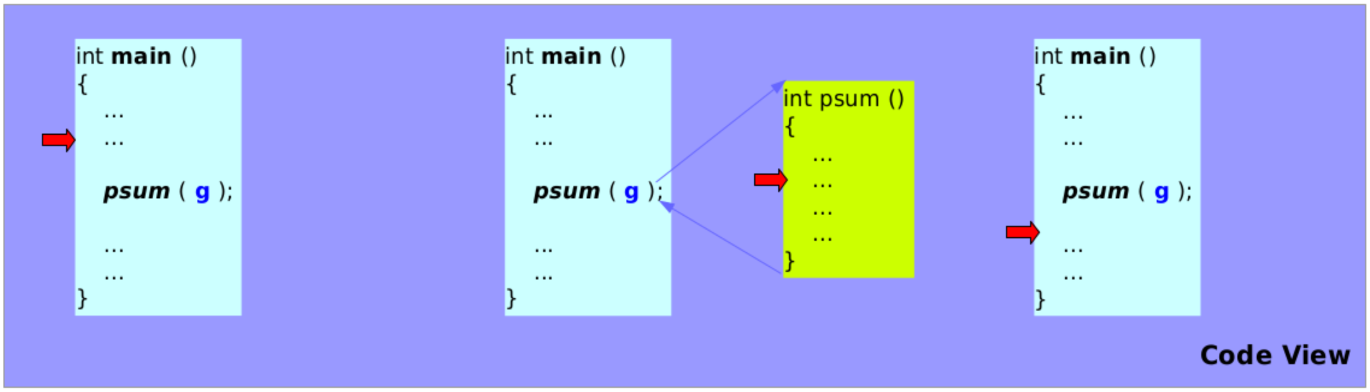
Extent (Life Time)

before the call to **psum**()

during **psum**() is being executed

after the call to **psum**()





```
int main ()
{
   ...
   ...

   psum ( g );

   ...
   ...
}
```

```
int main ()
{
   ...
   ...

   psum ( g );

   ...
   ...
}
```

```
int psum ()
{
   ...
   ...
   ...
   ...
}
```

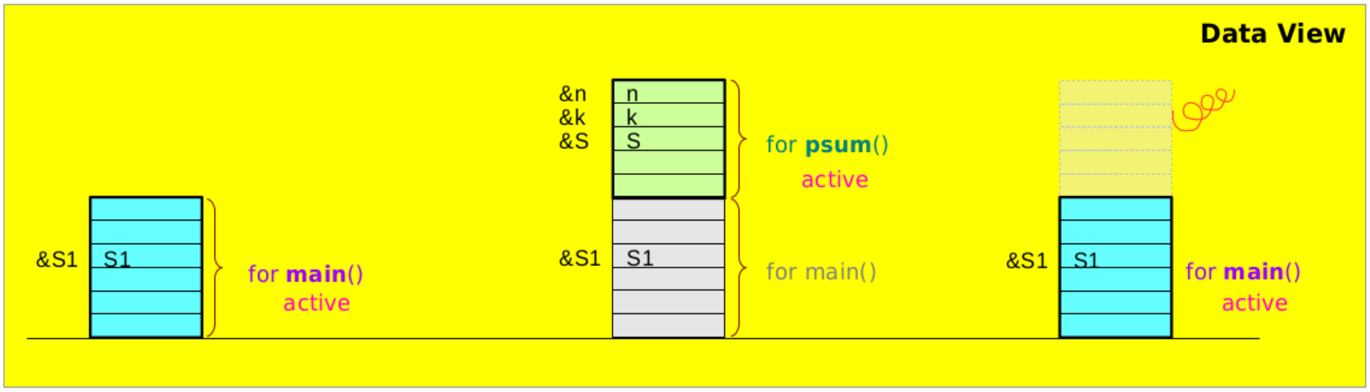
```
int main ()
{
   ...
   ...

   psum ( g );

   ...
   ...
}
```

Code View

before the call to **psum**()

during **psum**() is being executed

after the call to **psum**()



Data View
&n
&k
&S
n
k
S
for psum()
active
&S1
S1
for main()
active
&S1
S1
for main()
&S1
S1
for main()
active