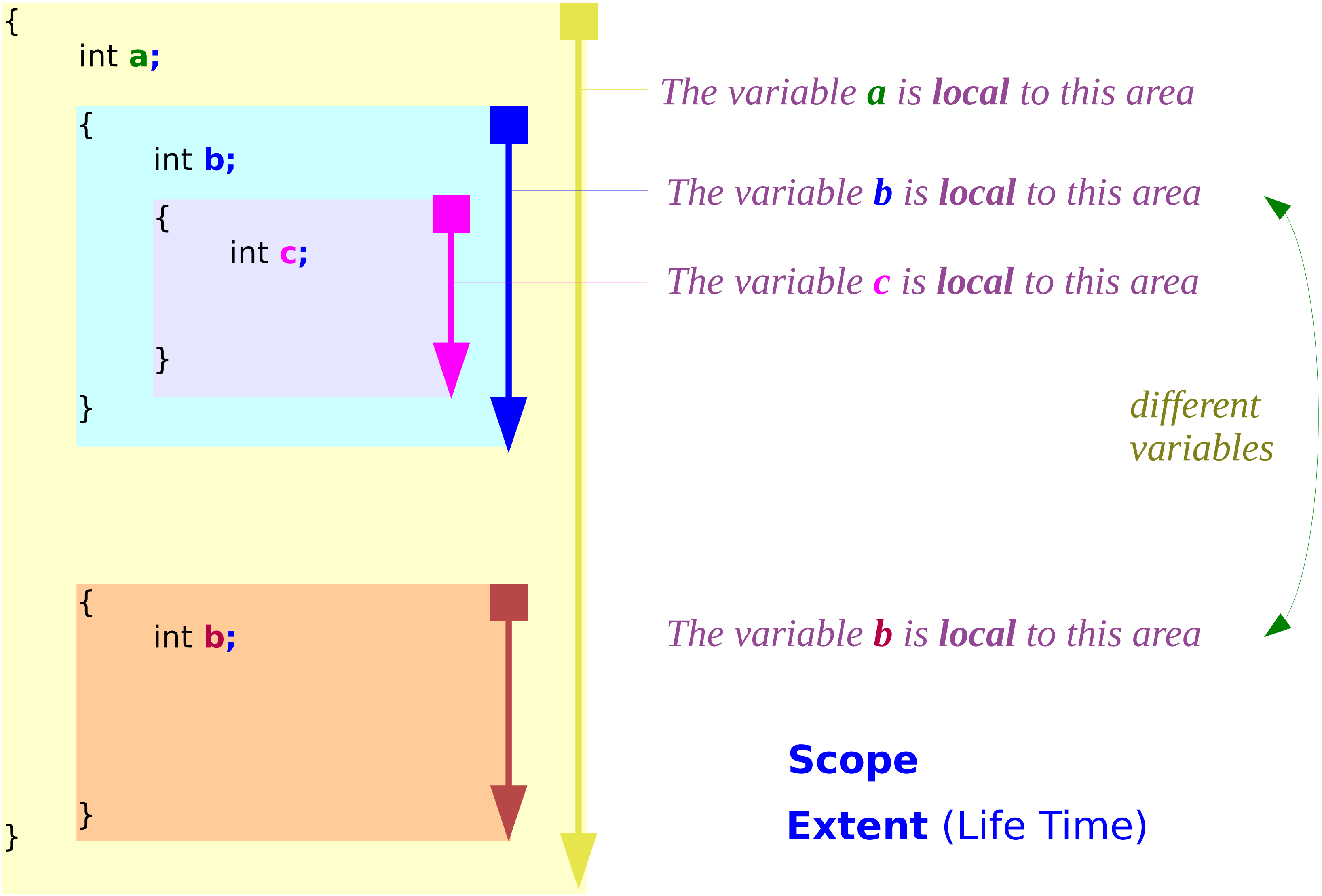
Functions & Variables (1A)

Young Won Lim
12/22/16

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using OpenOffice.

Scope



```
{
      int a;
      {
            int b;
            {
                  int c;

            }
      }


      {
            int b;



      }
}
```

*The variable **a** is **local** to this area*

*The variable **b** is **local** to this area*

*The variable **c** is **local** to this area*

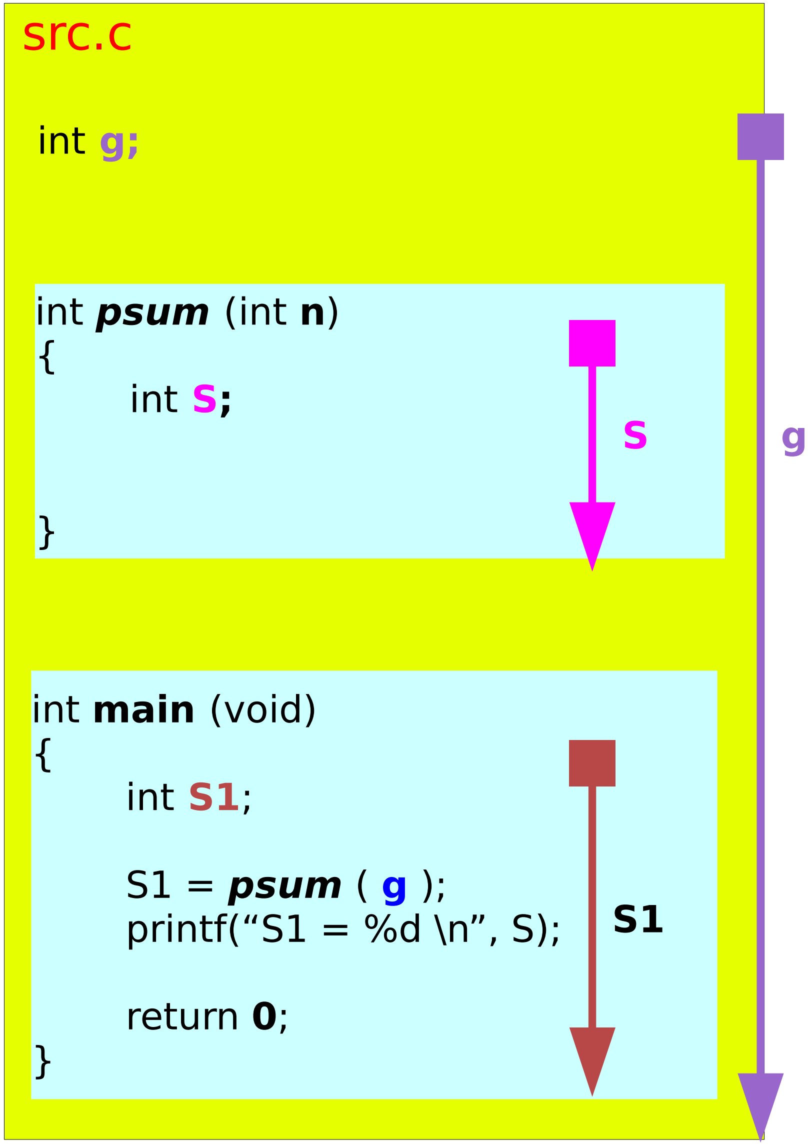
different variables

*The variable **b** is **local** to this area*

Scope

Extent (Life Time)

Global vs. Local Variables



```
int g;

int psum (int n)
{
        int S;


}


int main (void)
{
        int S1;

        S1 = psum ( g );
        printf(“S1 = %d \n”, S);

        return 0;
}
```

*The variable **g** is not **local** to any block*
It is defined outside of the main function
*It is therefore **global**.*

Assembly Code (Text)

src.c

```
Int g = 2;

int psum (int n)
{
        int k, S = 0;
        for (k=1; k<=n; ++k) S += k;
        return S;
}

int main (void)
{
        int S1;

        S1 = psum ( g );
        printf(“S1 = %d \n”, S);

        return 0;
}
```

```
psum.o:     file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <psum>:
   0:   55                      push   %rbp
   1:   48 89 e5                mov    %rsp,%rbp
   4:   89 7d ec                mov    %edi,-0x14(%rbp)
   7:   c7 45 fc 00 00 00 00    movl   $0x0,-0x4(%rbp)
   e:   c7 45 f8 01 00 00 00    movl   $0x1,-0x8(%rbp)
  15:   eb 0a                   jmp    21 <psum+0x21>
  17:   8b 45 f8                mov    -0x8(%rbp),%eax
  1a:   01 45 fc                add    %eax,-0x4(%rbp)
  1d:   83 45 f8 01             addl   $0x1,-0x8(%rbp)
  21:   8b 45 f8                mov    -0x8(%rbp),%eax
  24:   3b 45 ec                cmp    -0x14(%rbp),%eax
  27:   7e ee                   jle    17 <psum+0x17>
  29:   8b 45 fc                mov    -0x4(%rbp),%eax
  2c:   5d                      pop    %rbp
  2d:   c3                      retq
```

start address (to be relocated later)

```
S1 = psum(1);
   8:   bf 01 00 00 00          mov    $0x1,%edi
   d:   e8 00 00 00 00          callq  12 <main+0x12>
  12:   89 45 f4                mov    %eax,-0xc(%rbp)
```

go to where **psum** routine is

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;
}
```

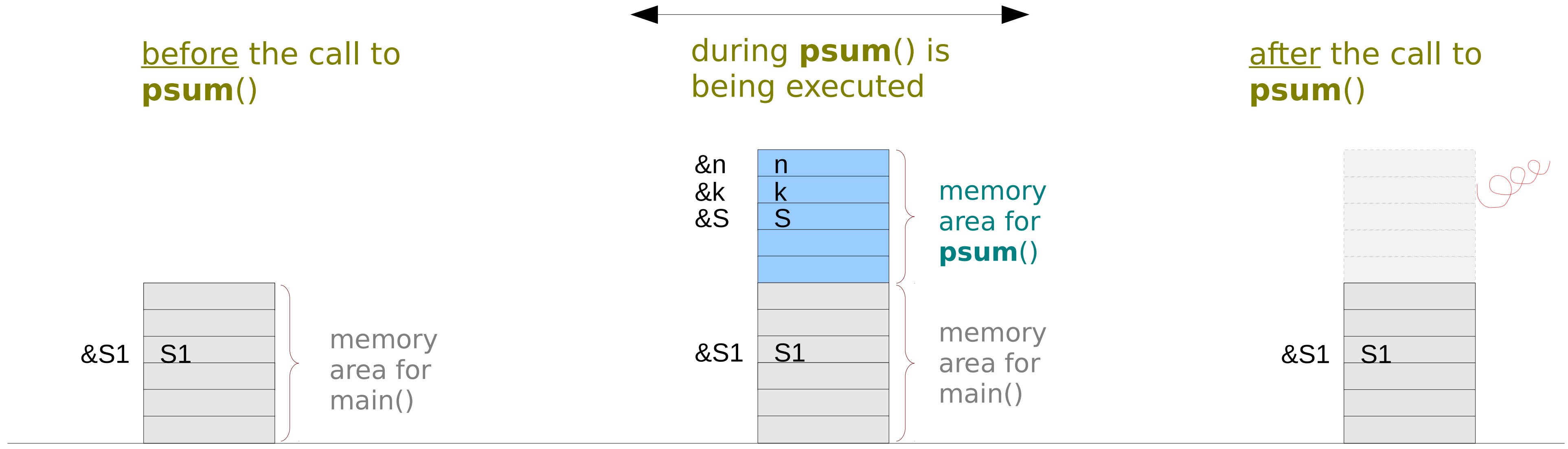
```
int main (void)
{
        int S1 = 0;

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

        return 0;
}
```

Extent (Life Time)

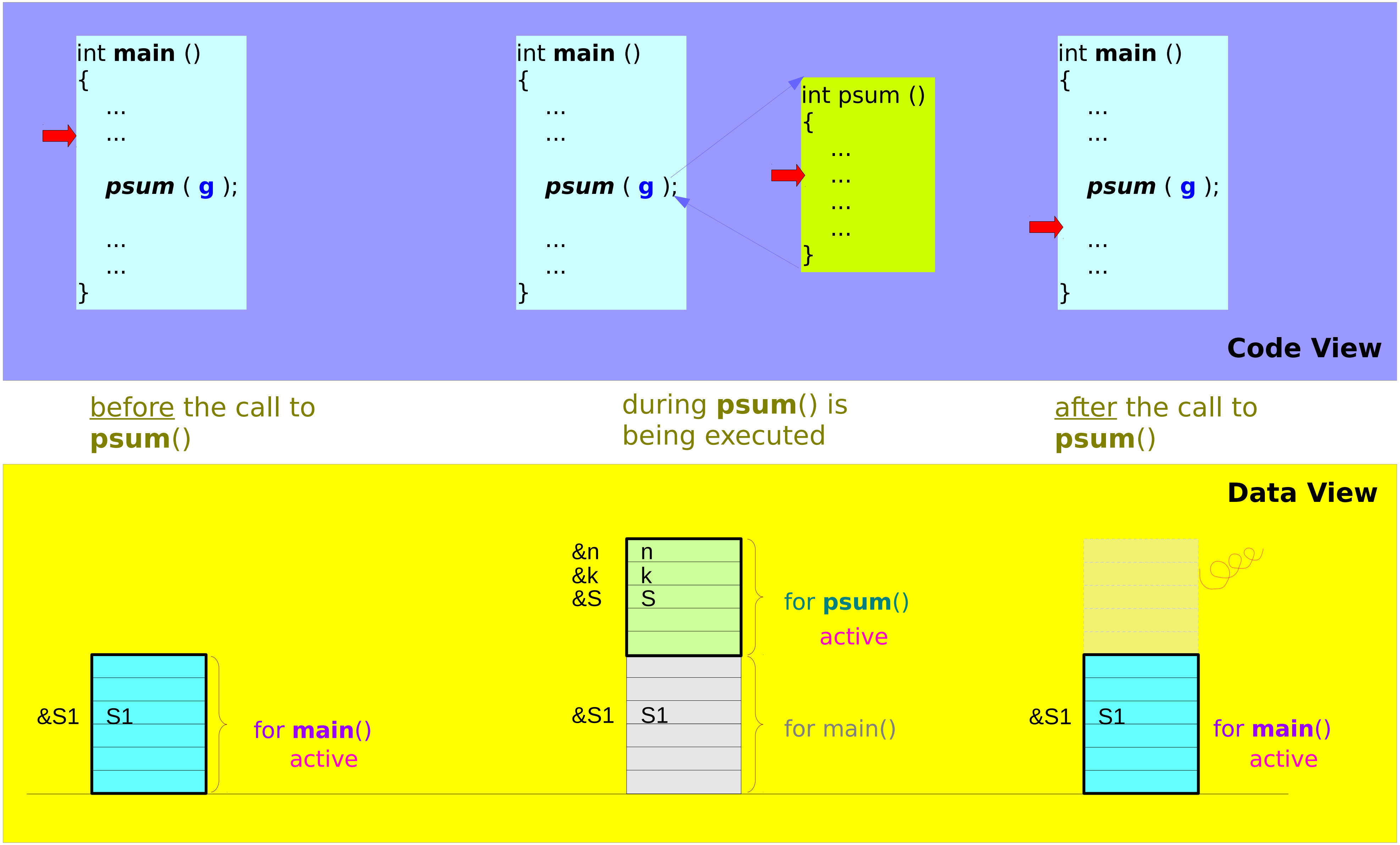
<u>before</u> the call to **psum**()

during **psum**() is being executed

<u>after</u> the call to **psum**()



Local Variables in a Stack Frame



```
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

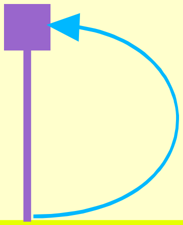
after the call to **psum**()

Data View

Static Variable

src.c

```
int psum (int n)
{
        static int S = 0;

        S += n;
       return (S);
}
```



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

        S1 = psum ( 1 );
        printf(“S1 = %d \n”, S1);
        S2 = psum ( 2 );
        printf(“S1 = %d \n”, S2);
        S3 = psum ( 3 );
        printf(“S1 = %d \n”, S3);

        return 0;
}
```

Static Specifier

psum *starts with S=0, ends with S=1* *1*

psum *starts with S=1, ends with S=3* *1+2*

psum *starts with S=3, ends with S=4* *1+2+3*

Function Prototypes

1. **no** parameters

int main (**void**)

2. a **fixed number** of parameters

int ***psum*** **(int n)**

int ***func*** **( )**

accepts a constant but unknown number of arguments

3. a **variable number** of parameters

int ***printf*** (const char *format **,...**)

one fixed parameter

variable number of parameters

but <u>must</u> have at least **one fixed** parameter for the use in the standard library (ISO C)

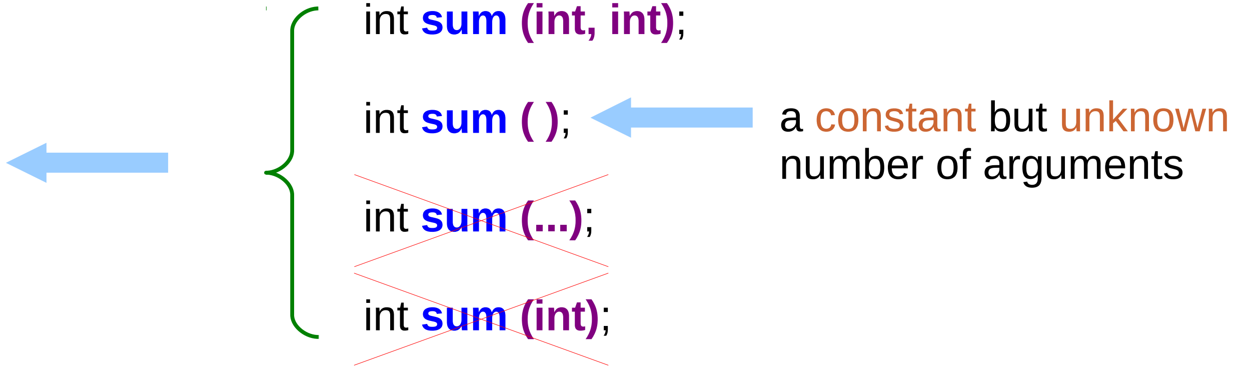
printf (“%d %d”, **a, b**);

int ***func*** **(...)** (not ISO C)

accepts a variable number of arguments

Function Prototypes Examples

```
#include <stdio.h>



void main(void)
{
   int a=10, b=20, c;

   c = sum(a, b);

   printf("c=%d \n", c);

}

int sum(int x, int y)
{
   return x + y;
}
```



int sum (int, int);

int sum ( );

a constant but unknown number of arguments

~~int sum (...);~~

~~int sum (int);~~

Function printf() Prototype

```
extern int printf (const char *__restrict __format, ...);
```

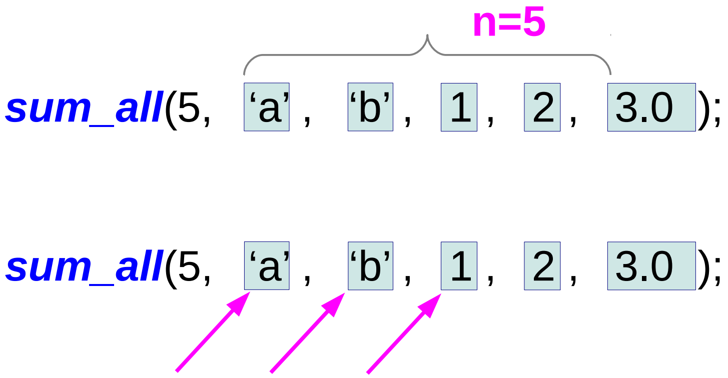
__restrict __: the reference is not aliased in the local context

```
int printf (const char *format, ...);
```

```
printf(“Hello, world!\n”);                    zero argument
printf(“m=%d\n”, m);                          one argument
printf(“x=%f, y=%f\n”, x, y);                 two arguments
printf(“a=%c, b=%c, c=%c\n”, a, b, c);        three arguments
```

Three Macros

va_list vl;	variable list definition
va_start (vl, n);	initialization with the number of elements
va_arg(vl, int);	pop each element in the list with the specified type
va_end(vl);	finalize the use of macros

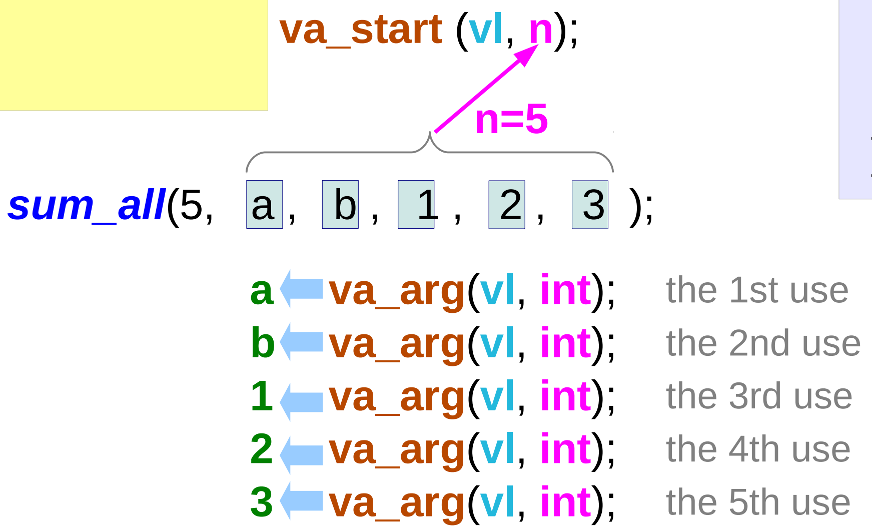

n=5
sum_all(5, 'a' , 'b' , 1 , 2 , 3.0);
sum_all(5, 'a' , 'b' , 1 , 2 , 3.0);

```
a =   va_arg(vl, int);       the 1st use
b =   va_arg(vl, int);       the 2nd use
1 =   va_arg(vl, int);       the 3rd use
2 =   va_arg(vl, int);       the 4th use
3.0 = va_arg(vl, double);    the 5th use
```

Variable Number of Arguments

```
int sum_all (int n, ...)
{
   int i, sum;

   va_list vl;
   va_start (vl, n);              // n elements
   sum = 0;
   for (i=0; i<n; ++i) {
     sum += va_arg(vl, int);      // take each int element
   }
   va_end(vl);

   return sum;
}
```

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

void sum_all (int, ...);

void main(void)
{
   int a=10, b=20, c;

   c = sum_all(2, a, b);
   printf("c=%d \n", c);

   c = sum_all(5, a, b, 1, 2, 3);
   printf("c=%d \n", c);

}
```



```
va_start (vl, n);

n=5

sum_all(5, a , b , 1 , 2 , 3 );

a ⇐ va_arg(vl, int);   the 1st use
b ⇐ va_arg(vl, int);   the 2nd use
1 ⇐ va_arg(vl, int);   the 3rd use
2 ⇐ va_arg(vl, int);   the 4th use
3 ⇐ va_arg(vl, int);   the 5th use
```

References

[1] Essential C, Nick Parlante
[2] Efficient C Programming, Mark A. Weiss
[3] C A Reference Manual, Samuel P. Harbison & Guy L. Steele Jr.
[4] C Language Express, I. K. Chun