Function Overview (1A)

Young Won Lim
10/30/17

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

This document was produced by using OpenOffice / LibreOffice.

3 Return 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= func1(100);
```

```
S= func2(100);

S= func2(100);
```

```
func3( 100 );
```

3 Return Types of Functions – Errors and Warnings

```
int func1( int a ) {

   a *= 999;

}
```

return val missing

```
int func2( int a ) {

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

}
```

return val missing

```
void func3( int a ) {

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

void : no return value
return; can be used

```
S= func1(100);

S= func1(100);
```

```
S= func2(100);

S= func2(100);
```

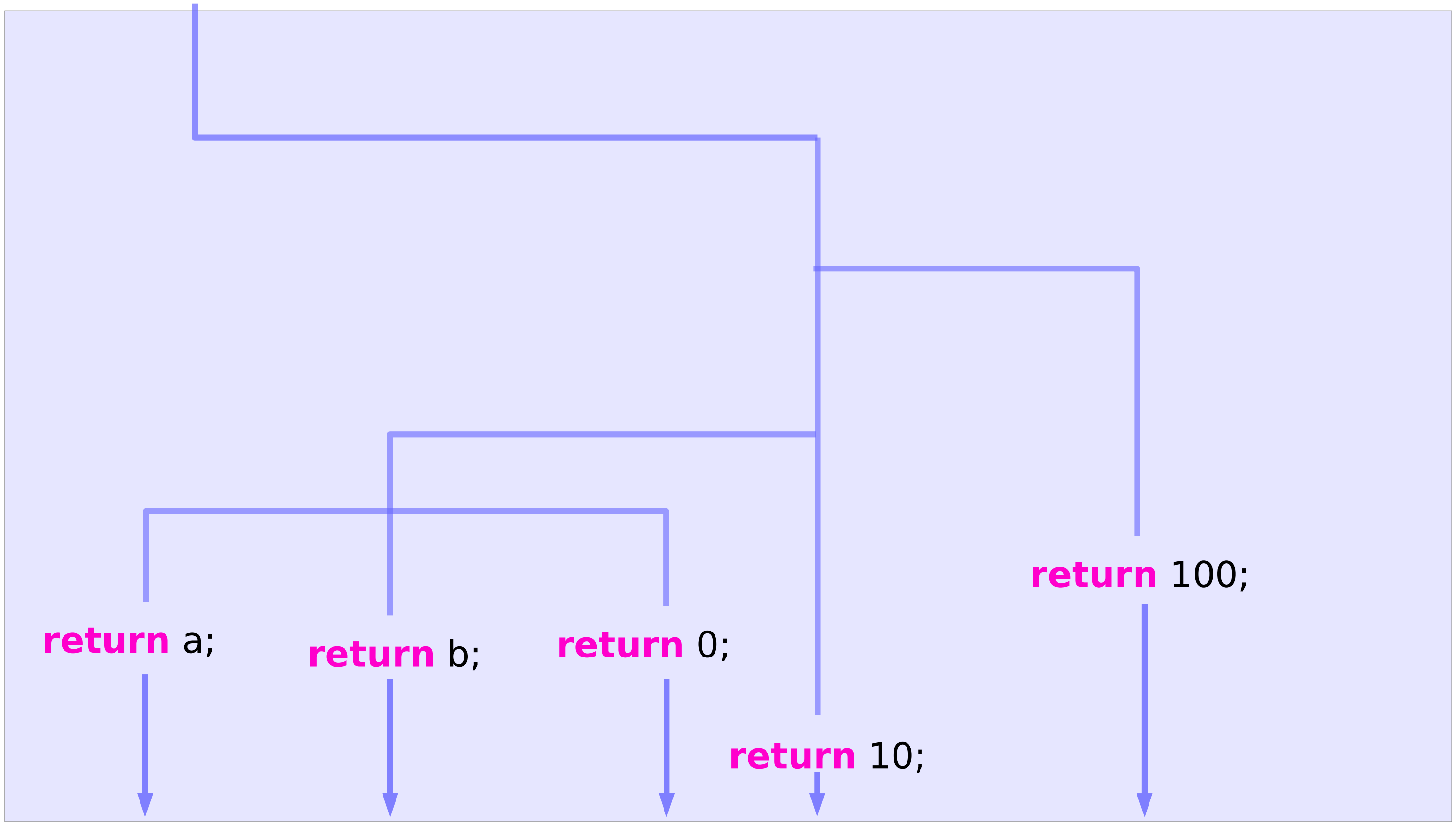
```
func3( 100 );
```

~~S= func3(100);~~

void returns no value
cannot assign a variable

Every branch must return an integer value

int func2(int a) {



}

2 Passing Types of Functions

```
void val_func( int a ) {
    X = a;  // input
    a = Y;  // meaningless
}
```

```
void ref_func( int *p ) {
    X = *p;  // input
    *p = Y;  // output
}
```

```
int m;
val_func( m );
```

```
int m;
ref_func( &m );
```

```
int *n;
val_func( *n );
```

```
int *n;
ref_func( n );
```

In-bound, Out-bound, and Bi-directional Parameters

in-bound only

```
void valf( int a ) {
        X = a;  // input
        a = Y;  // meaningless
}
```

bi-directional

```
void reff( int *p ) {
        X = *p;  // input
        *p = Y;  // output
}
```

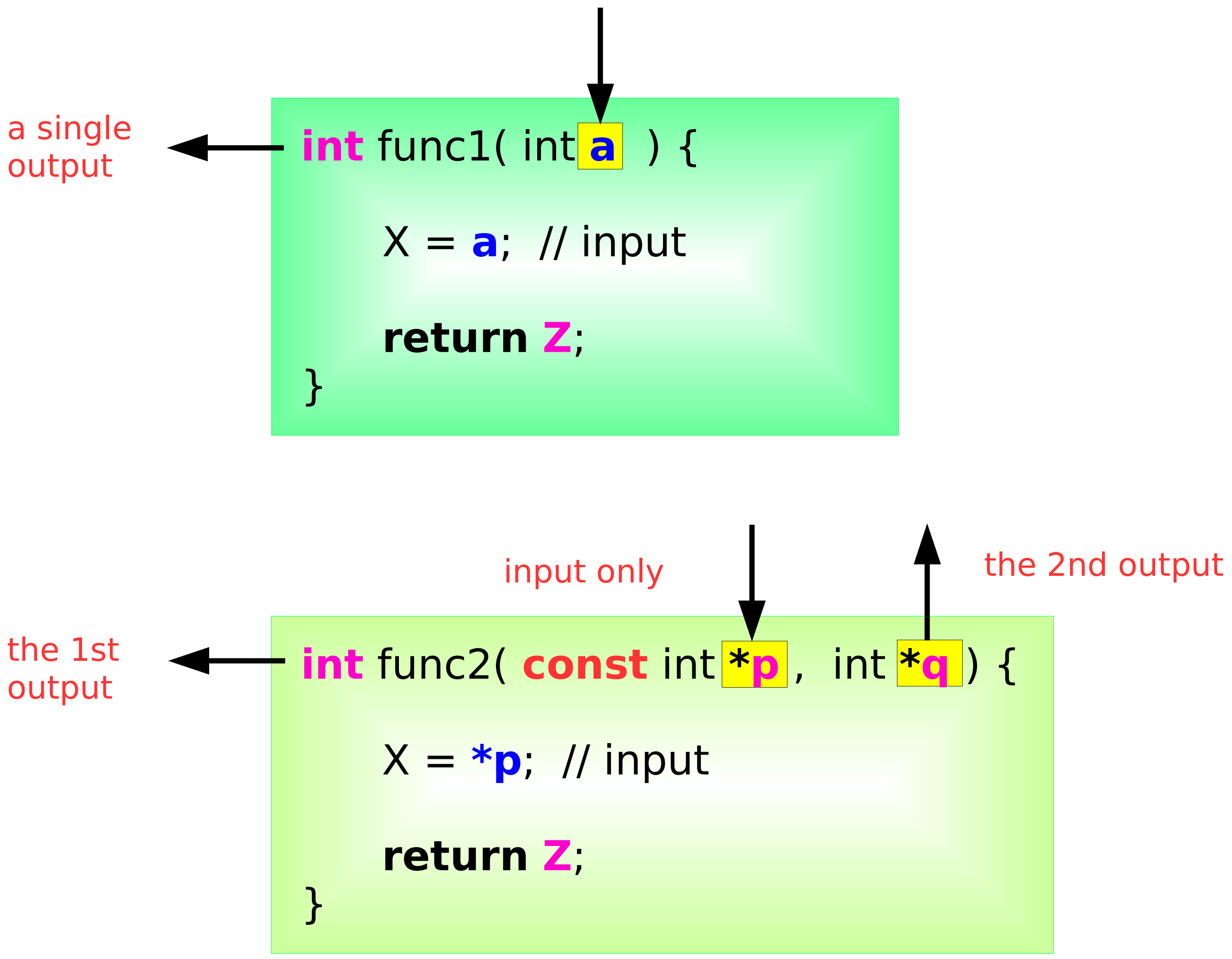
in-bound only

```
void reff( const int *p ) {
        X = *p;  // input
        ~~*p = Y;~~  // prohibited
}
```

out-bound

```
void reff(  int *p ) {

        *p = Y;  // output
}
```

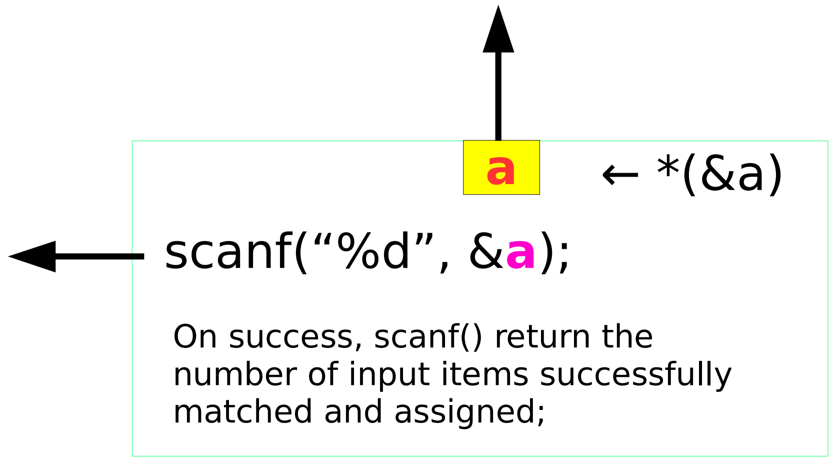
Extra Outputs (1)



```
int func1( int a  ) {

      X = a;  // input

      return Z;
}
```

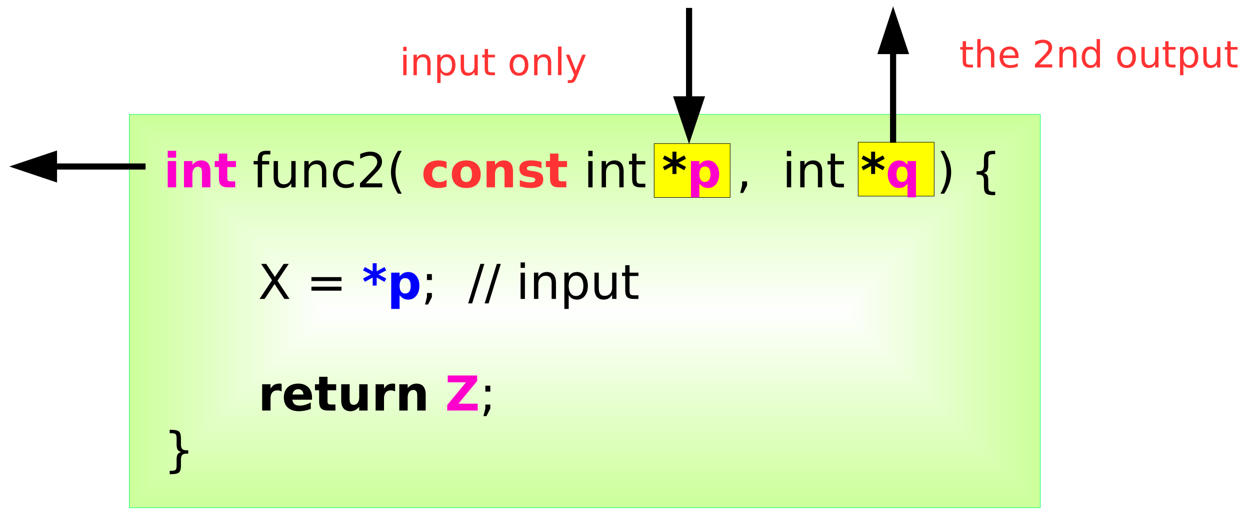
```
int func2( const int *p ,  int *q ) {

      X = *p;  // input

      return Z;
}
```

Extra Outputs (2)



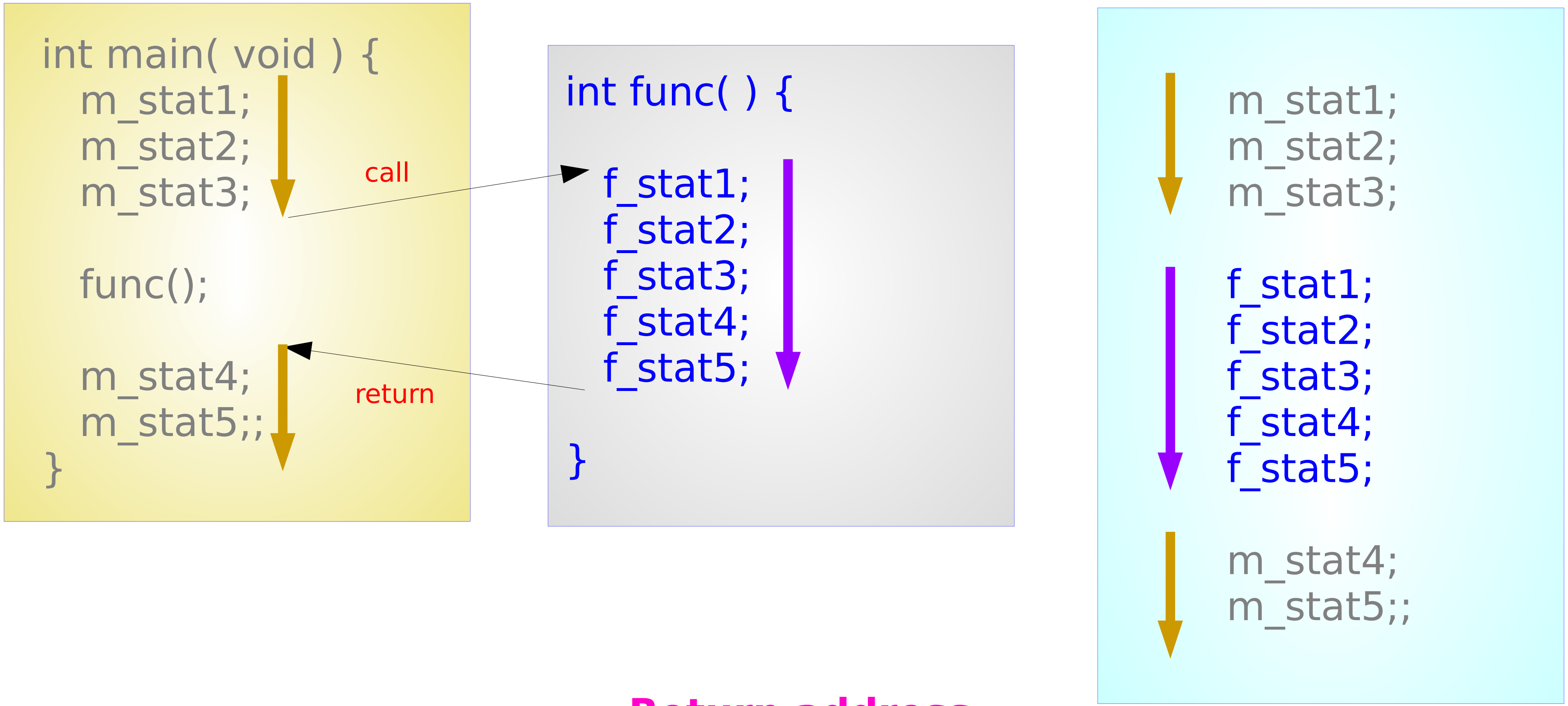
```
scanf("%d", &a);
```

On success, scanf() return the number of input items successfully matched and assigned;



```
int func2( const int *p ,  int *q ) {

      X = *p;  // input

      return Z;
}
```

Function Calls and Control Transfers



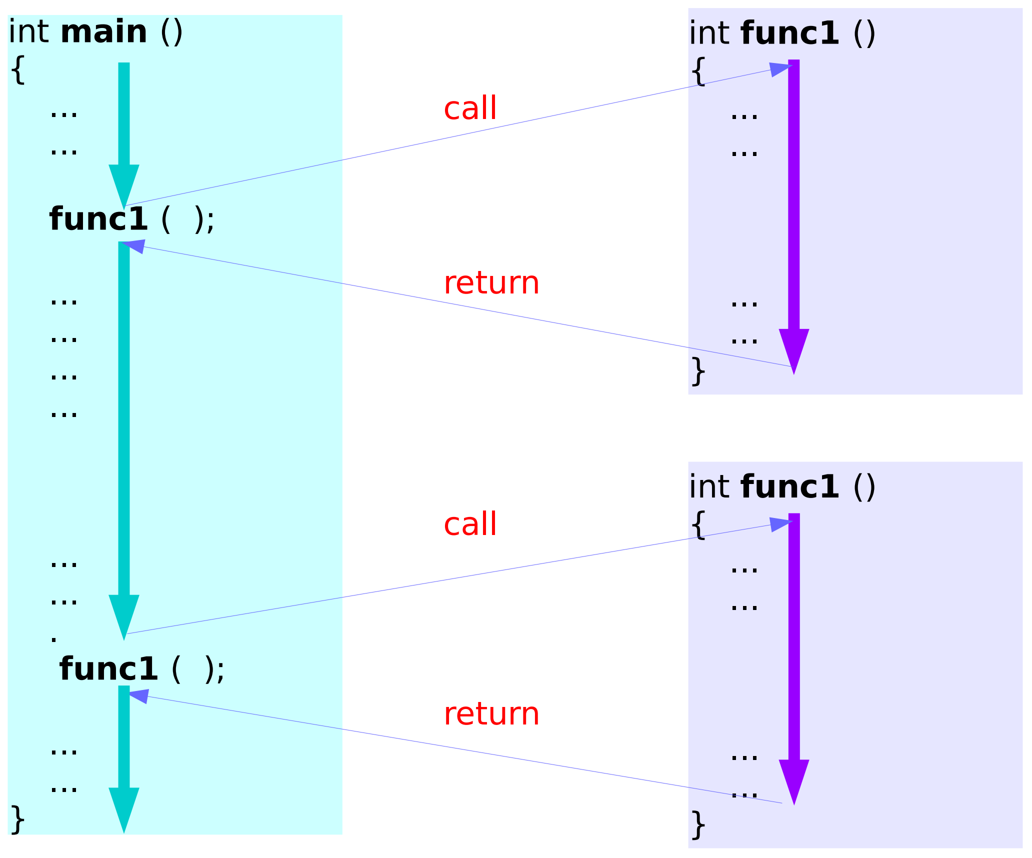
```
int main( void ) {
   m_stat1;
   m_stat2;
   m_stat3;

   func();

   m_stat4;
   m_stat5;;
}
```

```
int func( ) {

   f_stat1;
   f_stat2;
   f_stat3;
   f_stat4;
   f_stat5;

}
```

```
m_stat1;
m_stat2;
m_stat3;

f_stat1;
f_stat2;
f_stat3;
f_stat4;
f_stat5;

m_stat4;
m_stat5;;
```

Return address
Local variables

Return Addresses

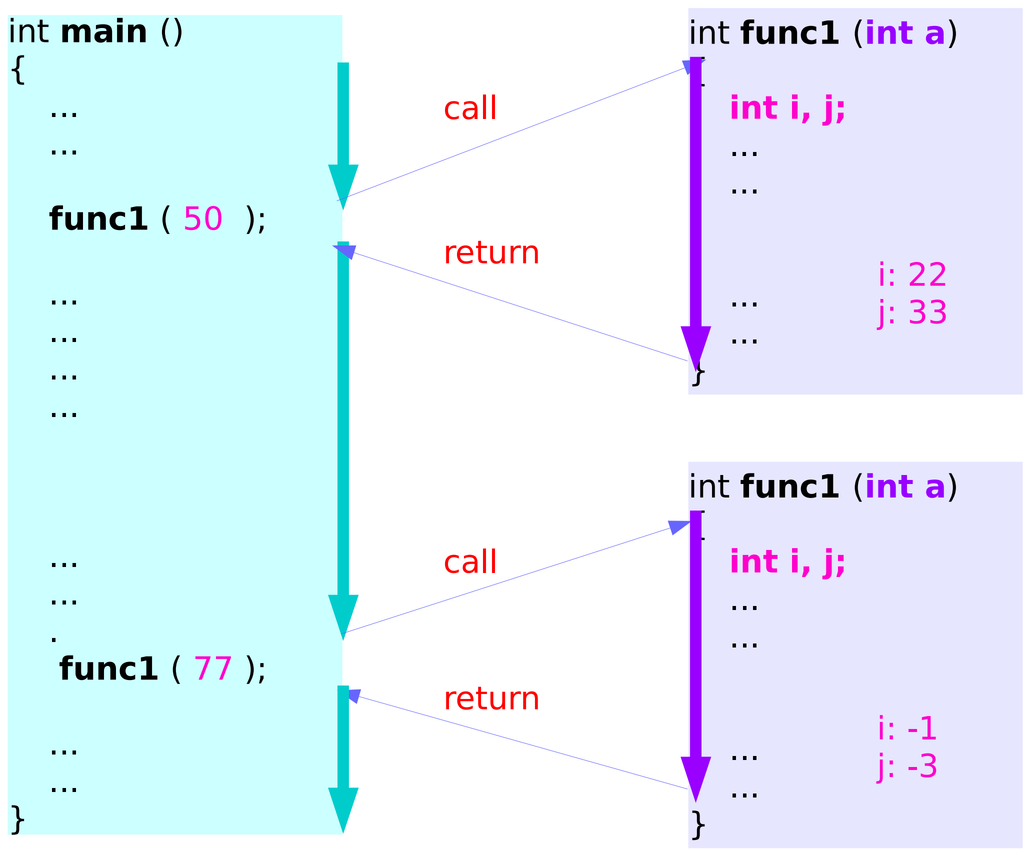


each invocation of the same function can have different return addresses

each invocation, its own return address

Return address

Local Variables



```
int main ()
{
   ...
   ...
   func1 ( 50  );
   ...
   ...
   ...
   ...
   ...
   ...
   .
    func1 ( 77 );
   ...
   ...
}
```

```
int func1 (int a)
{
   int i, j;
   ...
   ...
   ...
   ...
}
```

```
int func1 (int a)
{
   int i, j;
   ...
   ...
   ...
   ...
}
```

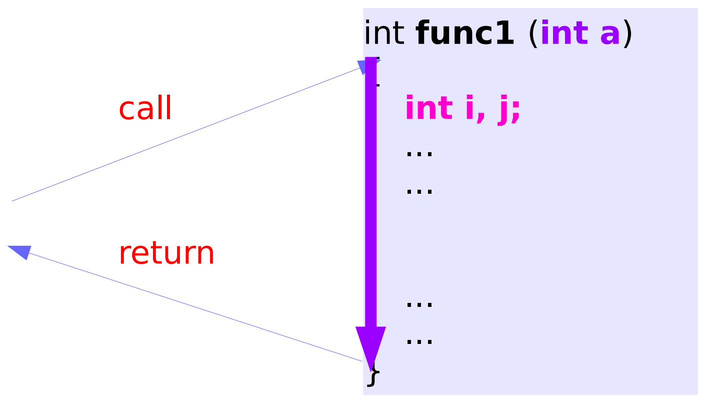
in each invocation of the same function the local variables usually have different values

each invocation, its own local variables

these local variables are must be preserved until the function returns (while the function is active)

Local variables

Stack Data Structure

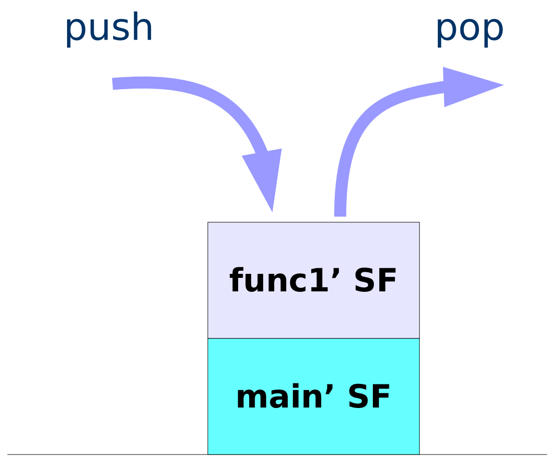


```
int func1 (int a)
{
    int i, j;
    …
    …


    …
    …
}
```

from the beginning and
to the end of a function call
(while the function is active)

- its return address
- its local variables

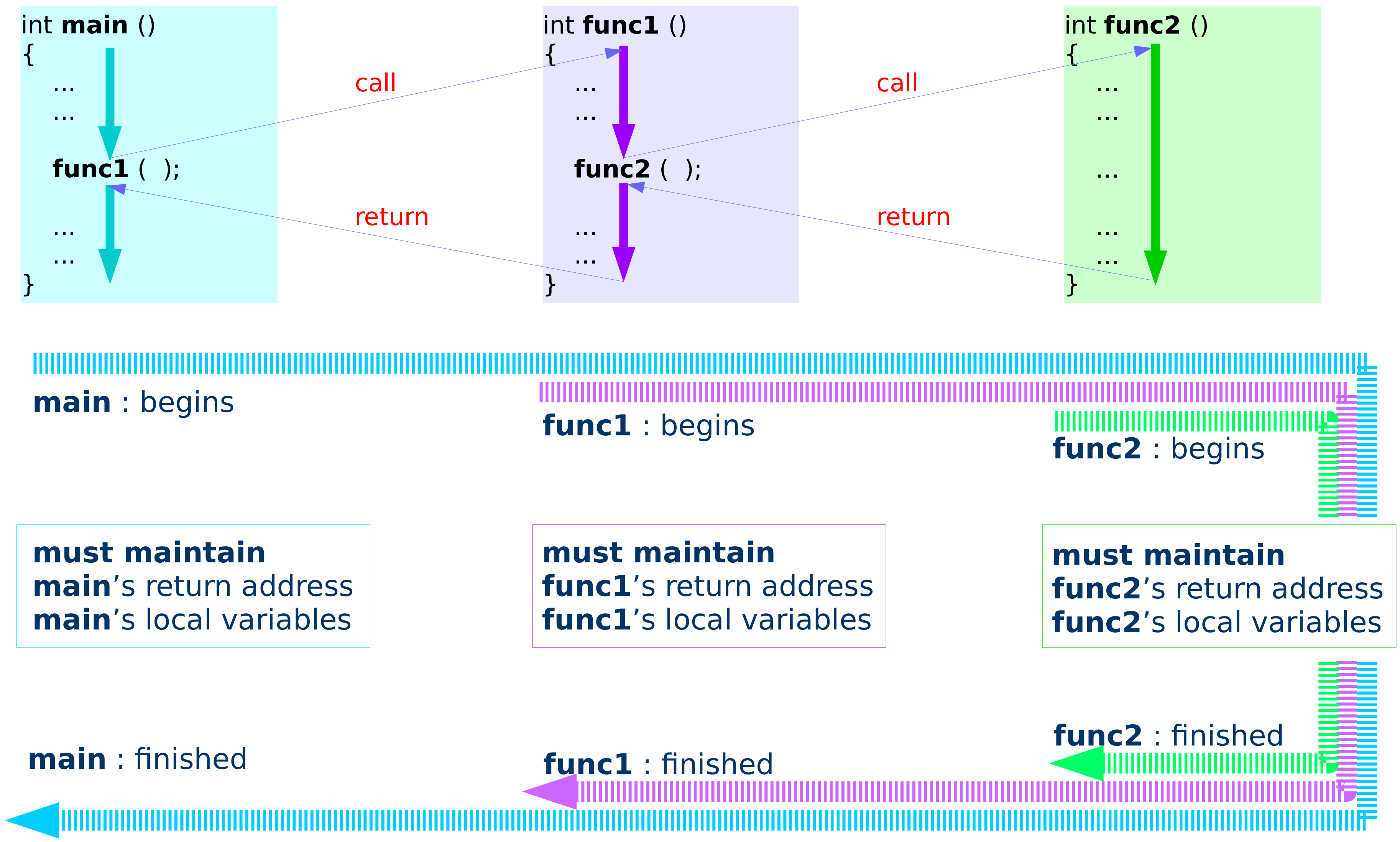
must be preserved



each function has its own Stack Frame
where each function store its own
return address and local variables

Stack Data Structure
(Last In First Out)

Nested function calls

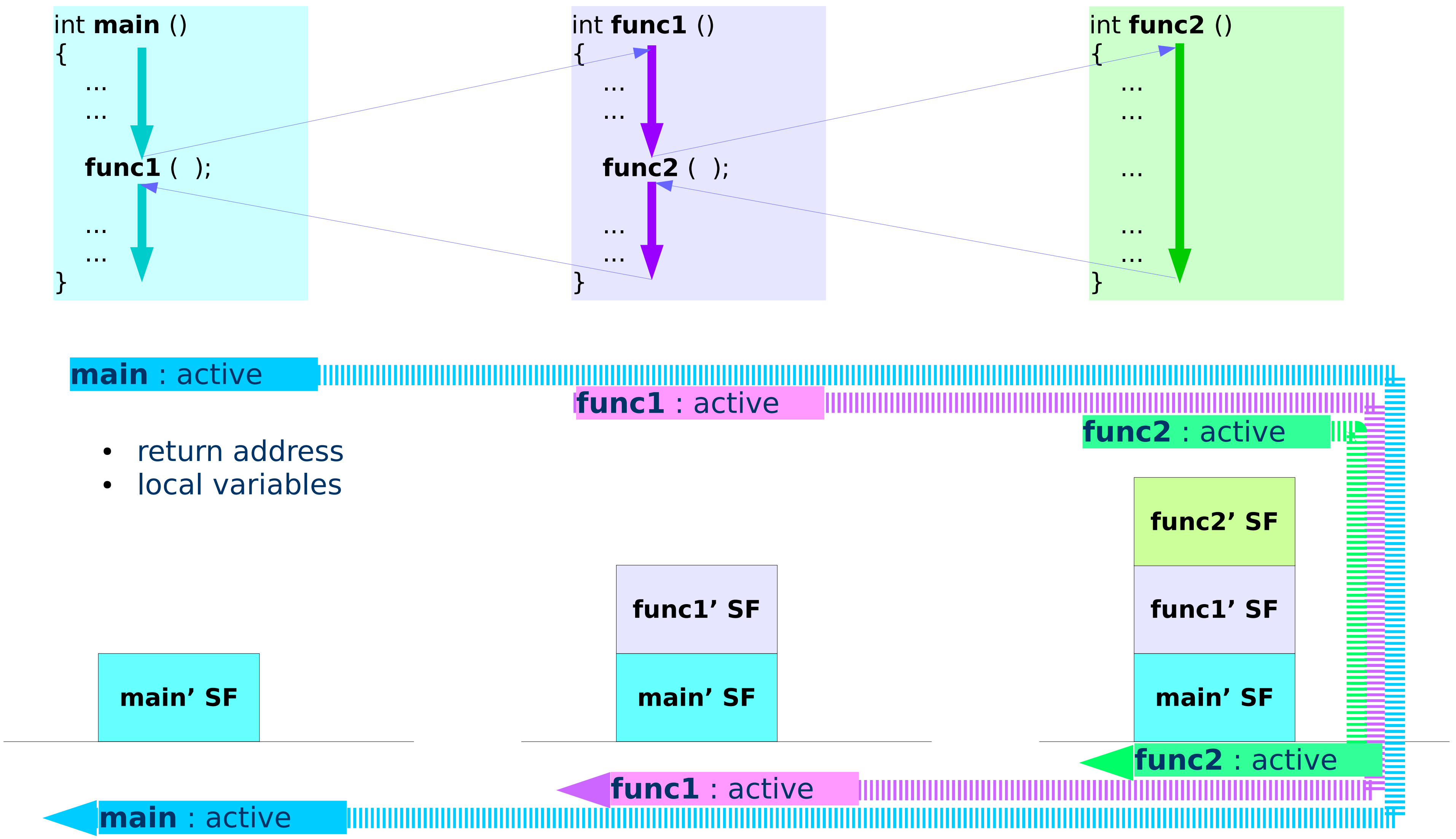


must maintain
main's return address
main's local variables

must maintain
func1's return address
func1's local variables

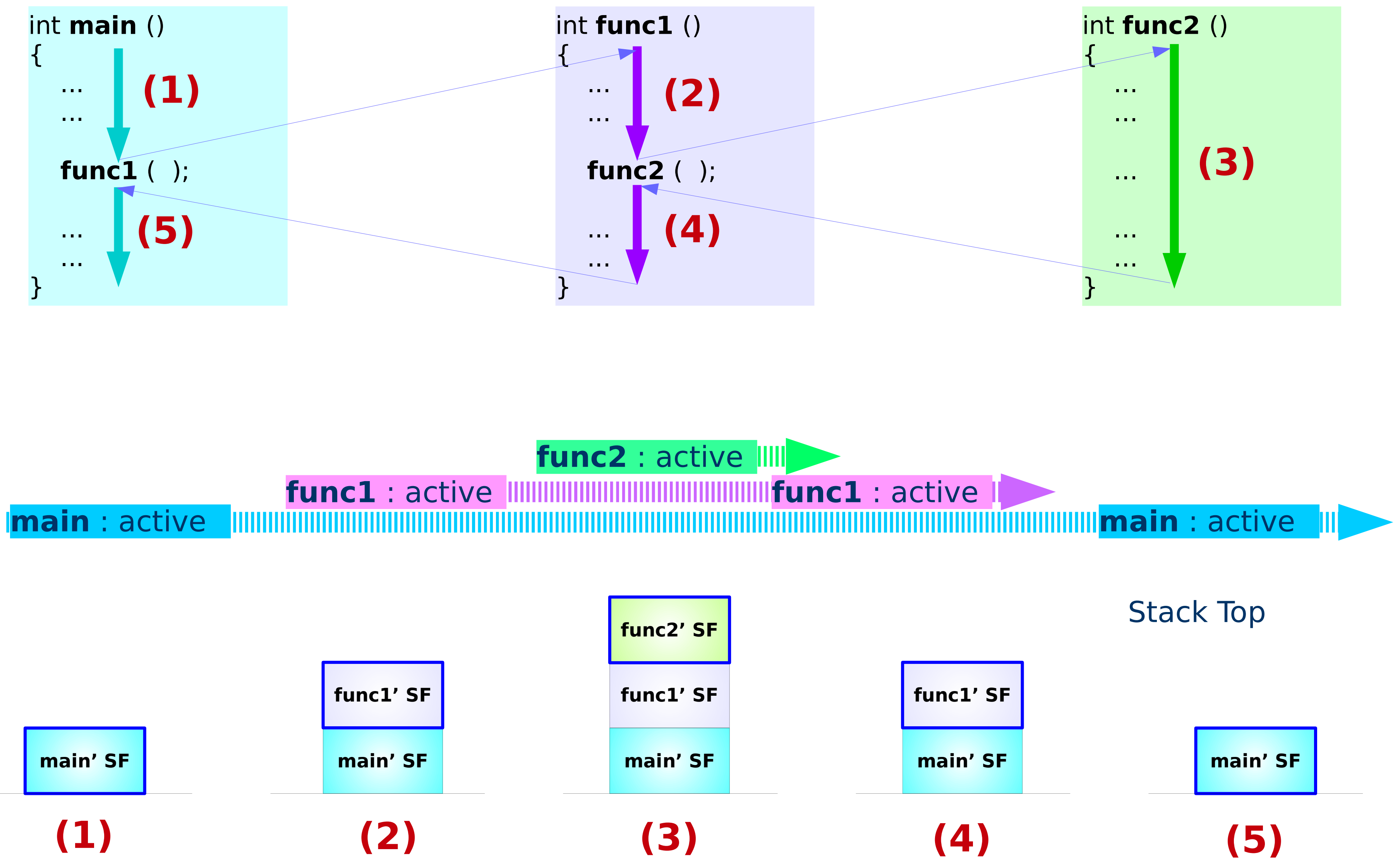
must maintain
func2's return address
func2's local variables

Nested function calls and stack frames

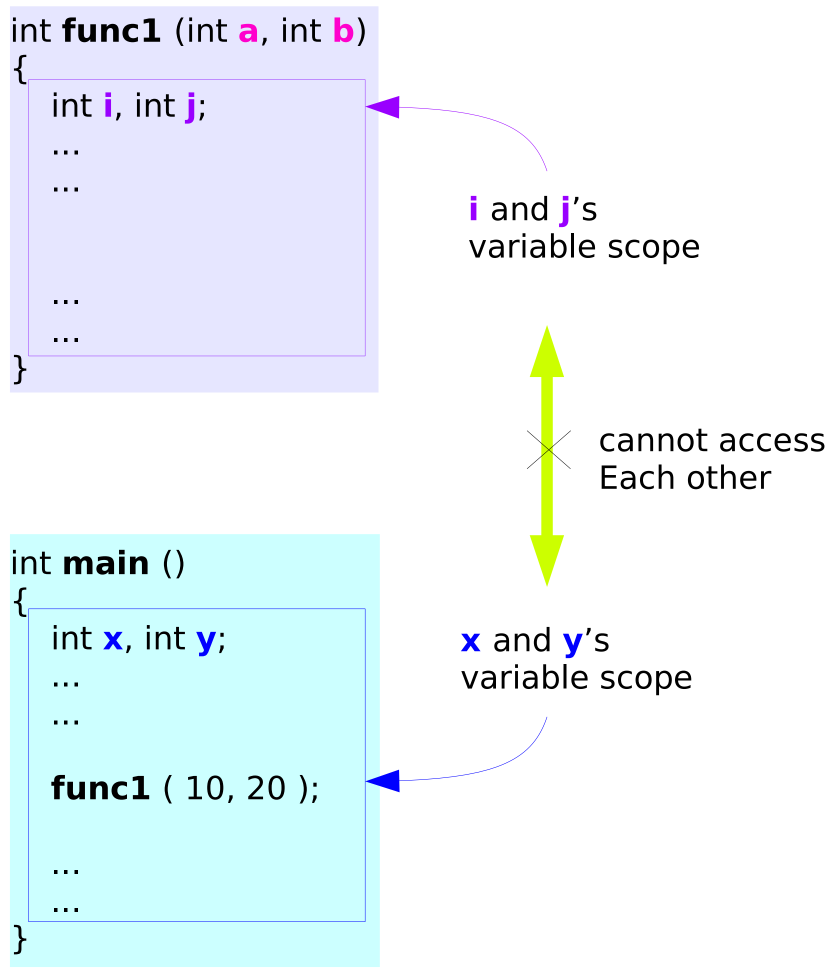


- return address
- local variables

Storage Duration



Variable Scopes

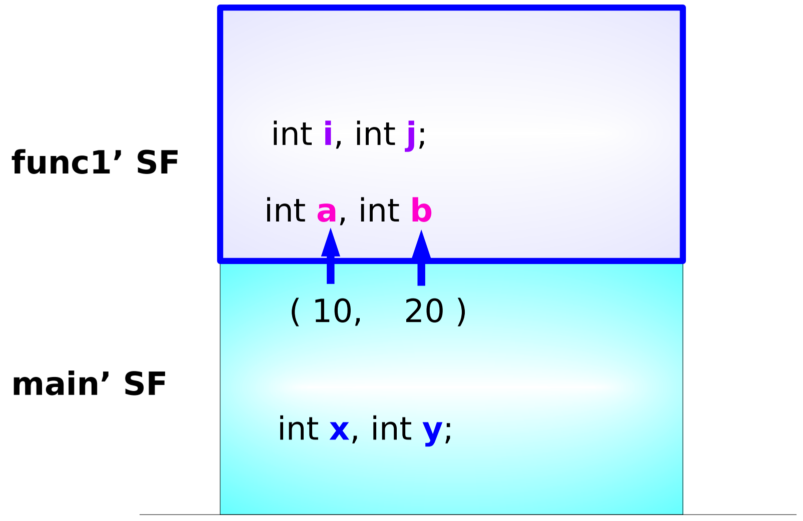


```
int func1 (int a, int b)
{
    int i, int j;
    …
    …


    …
    …
}
```

```
int main ()
{
    int x, int y;
    …
    …

    func1 ( 10, 20 );

    …
    …
}
```

Only **top** stack frame is active
and its variable can be accessed

Communications are performed
only through the **parameter** variables



Task: Finding 3 Partial Sums

$$S_n = \sum_{k=1}^{n} k$$

$$S_1 = \sum_{k=1}^{1} k$$

$$S_2 = \sum_{k=1}^{2} k$$

$$S_3 = \sum_{k=1}^{3} k$$

$S_1 = 1$

```
printf("S1 = %d \n", S1);
```

$S_2 = 1 + 2$

```
printf("S2 = %d \n", S2);
```

$S_3 = 1 + 2 + 3$

```
printf("S3 = %d \n", S3);
```

Finding 3 Partial Sums – 3 for loops

$$S_1 = \sum_{k=1}^{\textcircled{1}} k = 1$$

```
S1 = 0;
for (k=1; k<=1; ++k) S1 += k;
```

```
printf("S1 = %d \n", S1);
```

$$S_2 = \sum_{k=1}^{\textcircled{2}} k = 1 + 2$$

```
S2 = 0;
for (k=1; k<=2; ++k) S2 += k;
```

```
printf("S2 = %d \n", S2);
```

$$S_3 = \sum_{k=1}^{\textcircled{3}} k = 1 + 2 + 3$$

```
S3 = 0;
for (k=1; k<=3; ++k) S3 += k;
```

```
printf("S3 = %d \n", S3);
```

3 blocks with local variables

```
1  ⇨ n;
{      // block 1
       int n   ⇦;
       int k, S = 0;
       for (k=1; k<=n; ++k) S += k;
}
S1 ⇦  S;
```

```
printf(“S1 = %d \n”, S1);
```

```
2  ⇨ n;
{      // block 2
       int n   ⇦;
       int k, S = 0;
       for (k=1; k<=n; ++k) S += k;
}
S2 ⇦  S;
```

```
printf(“S2 = %d \n”, S2);
```

```
3  ⇨ n;
{      // block 3
       int n   ⇦;
       int k, S = 0;
       for (k=1; k<=n; ++k) S += k;
}
S3 ⇦  S;
```

```
printf(“S3 = %d \n”, S3);
```

Local Variables

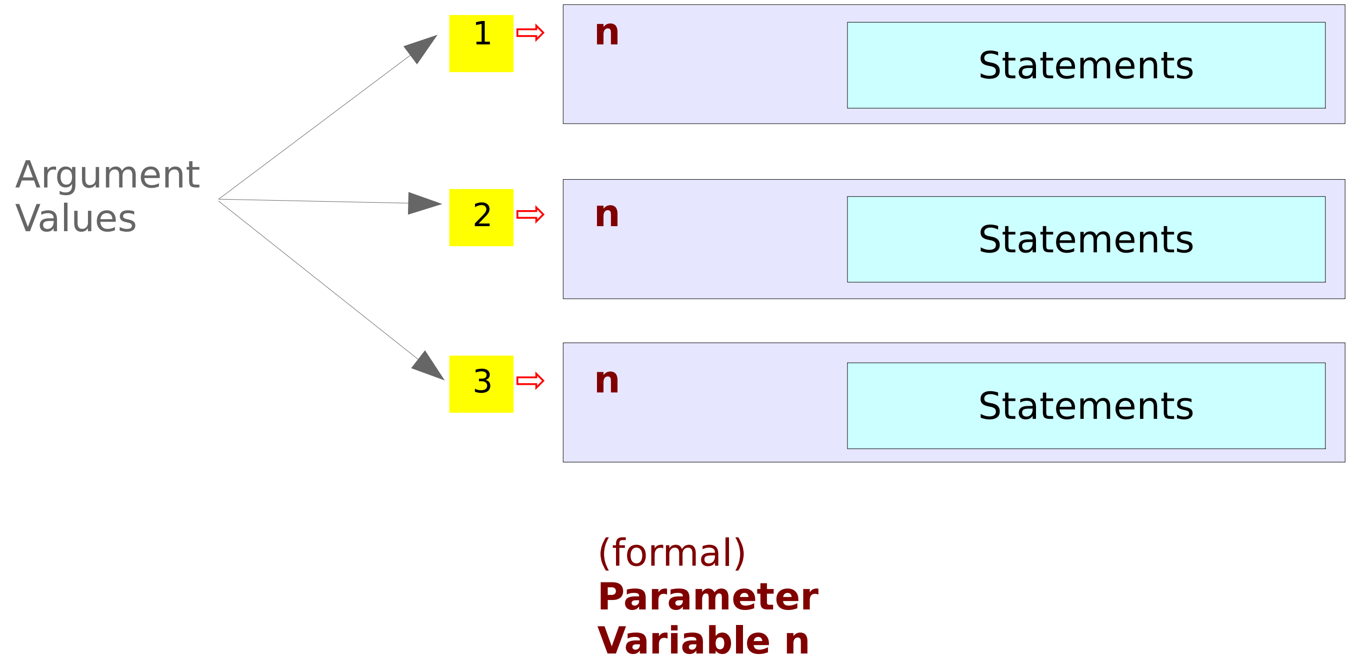
```
int n ;
int k, S = 0;
```

X 3

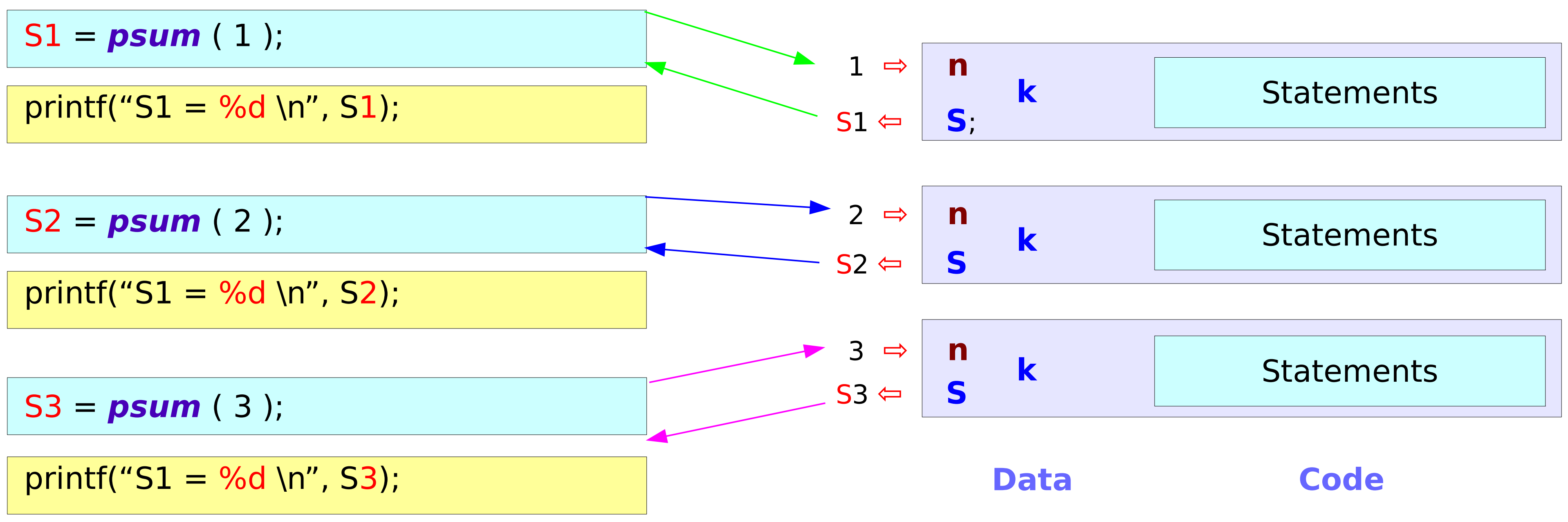
the same named variables
with different values

Arguments and Parameters

```
S1 = psum ( 1 );
S2 = psum ( 2 );
S2 = psum ( 3 );
```



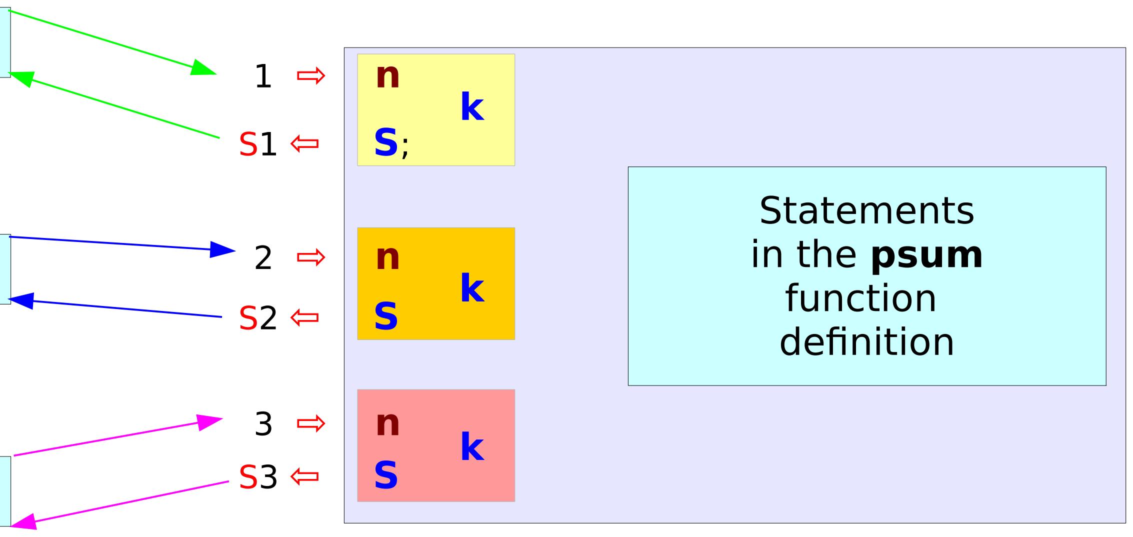
Data and Code

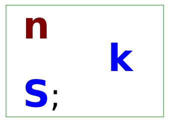


Data	**Code**
different local variables	3 copies of statements
activation	**redundancy**
resides <u>stack</u> memory area	only <u>one</u> set resides on the memory

Local Variables

```
S1 = psum ( 1 );

printf("S1 = %d \n", S1);

S2 = psum ( 2 );

printf("S1 = %d \n", S2);

S3 = psum ( 3 );

printf("S1 = %d \n", S3);
```





Active only when **psum**() is called

Function Prototype and Definition

```
■ ⇨ ;
{
      int n = input_n;
      int k, S = 0;
      for (k=1; k<=n; ++k) S += k;
}
⇦  S;
```

Function Prototype

```
⇦ int psum (int n) ;
```

Function Definition

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

Function Prototype & Definition in a File

src1.c

```
int psum (int n)  ;

int main (void)
{
        int S1, S2, S3;

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

        return 0;
}

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

To inform the compiler that ***psum*** *is the* ***name of a function*** *which has one integer type input and whose output type is integer*

Since ***psum*** *identifier is* ***declared****,* ***psum*** *can be* ***used*** *here.*

What the function ***psum*** *actually does is* ***defined*** *here.*

```
gcc -o run src1.c

./run
```

Only Function Definition in a File

src2.c

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

int main (void)
{
        int S1, S2, S3;

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

        return 0;
}
```

*The function **psum** is defined here.*

*Since **psum** identifier is declared (actually the function is defined), **psum** can be **used** here.*

```
gcc -o run src2.c

./run
```

One File Examples

src1.c

```
int psum (int n)  ;

int main (void)
{
        int S1, S2, S3;

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

        return 0;
}

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

src2.c

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

int main (void)
{
        int S1, S2, S3;

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

        return 0;
}
```

Two File Examples

src3.c

```
int psum (int n)  ;

int main (void)
{
        int S1, S2, S3;

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

        return 0;
}
```

src4.c

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

$$\begin{tikzcd}
\texttt{gcc -c src3.c} \arrow[r] & \texttt{src3.o}
\end{tikzcd}$$

$$\begin{tikzcd}
\texttt{gcc -c src4.c} \arrow[r] & \texttt{src4.o}
\end{tikzcd}$$

```
gcc -o run src3.o src4.o

./run
```

Header File Examples

src5.h

```
int psum (int n)  ;
```

src5.c

```
#include "src4.h"

int main (void)
{
        int S1, S2, S3;

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

        return 0;
}
```

src6.c

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

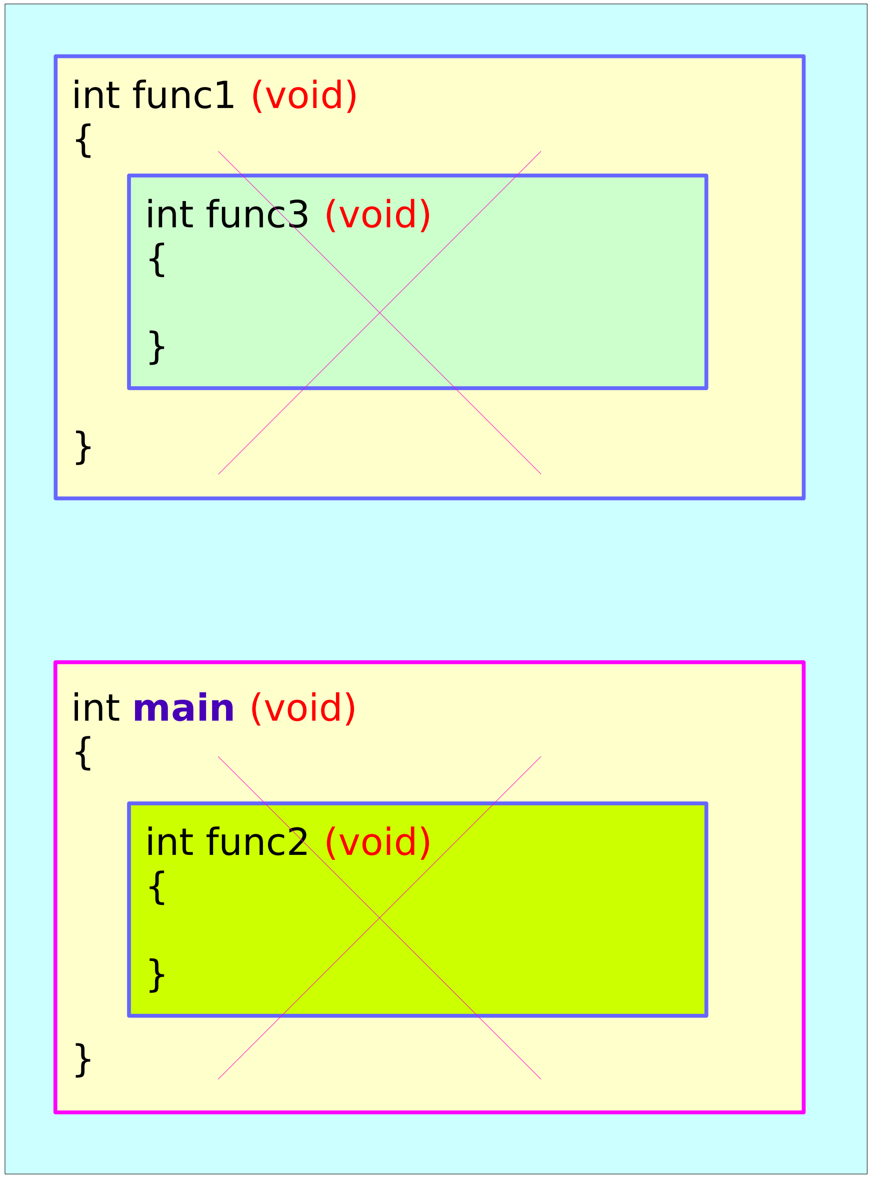
```
gcc -c src5.c   ➡   src5.o

gcc -c src6.c   ➡   src6.o

gcc -o run src5.o src6.o

./run
```

Function Definitions (1)

```
int func1 (void)
{


}

int func2 (void)
{


}

int main (void)
{


}
```

functions are defined
outside the main function

Function Definitions (2)



Nested function definitions
are not allowed

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