Functions

Basics of Programming 1

DEPARTMENT OF
NETWORKED SYSTEMS
AND SERVICES

G. Horváth, A.B. Nagy, Z. Zsóka, P. Fiala, A. Vitéz

2 October, 2024

Content

Chapter 1

Functions

Segmentation – motivation

Let's create a program, that prints out the sum of the squares of all positive numbers, that are smaller than 12! $(1^2 + 2^2 + \cdots + 11^2)$

Segmentation – motivation

Let's create a program, that prints out the sum of the squares of all positive numbers, that are smaller than 12! $(1^2 + 2^2 + \cdots + 11^2)$

```c
#include <stdio.h> /* for printf */

int main(void)
{
  int i, sum; /* aux. variable and sum of squares*/

  sum = 0;                          /* initialization */
  for (i = 1; i < 12; i = i+1) /* i = 1,2,...,11 */
    sum = sum + i*i;                /* summing */

  printf("The square sum: %d\n", sum);
  return 0;
}
```

link

Segmentation – motivation

```c
int main(void) {
  int i, sum1, sum2, sum3;

  sum1 = 0;            /* for 12 */
  for (i = 1; i < 12; i = i+1)
    sum1 = sum1 + i*i;

  sum2 = 0;            /* for 24 */
  for (i = 1; i < 24; i = i+1)
    sum2 = sum2 + i*i;

  sum3 = 0;            /* for 30 */
  for (i = 1; i < 30; i = i+1)
    sum3 = sum3 + i*i;

  printf("%d, %d, %d\n",
    sum1, sum2, sum3);
  return 0;
}
```

link

Let's create a program, that will perform the previous tasks with numbers 12, 24 and 30!

Segmentation – motivation

```c
int main(void) {
  int i, sum1, sum2, sum3;

  sum1 = 0;            /* for 12 */
  for (i = 1; i < 12; i = i+1)
    sum1 = sum1 + i*i;

  sum2 = 0;            /* for 24 */
  for (i = 1; i < 24; i = i+1)
    sum2 = sum2 + i*i;

  sum3 = 0;            /* for 30 */
  for (i = 1; i < 30; i = i+1)
    sum3 = sum3 + i*i;

  printf("%d, %d, %d\n",
    sum1, sum2, sum3);
  return 0;
}
```

link

Let's create a program, that will perform the previous tasks with numbers 12, 24 and 30!

Our solution

- was made by Copy+Paste+correct

Segmentation – motivation

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```c
int main(void) {
  int i, sum1, sum2, sum3;

  sum1 = 0;          /* for 12 */
  for (i = 1; i < 12; i = i+1)
    sum1 = sum1 + i*i;

  sum2 = 0;          /* for 24 */
  for (i = 1; i < 24; i = i+1)
    sum2 = sum2 + i*i;

  sum3 = 0;          /* for 30 */
  for (i = 1; i < 30; i = i+1)
    sum3 = sum3 + i*i;

  printf("%d, %d, %d\n",
    sum1, sum2, sum3);
  return 0;
}
```

link

Let's create a program, that will perform the previous tasks with numbers 12, 24 and 30!

Our solution

- was made by Copy+Paste+correct
- many possibilities for mistakes, errors

Segmentation – motivation

```c
int main(void) {
  int i, sum1, sum2, sum3;

  sum1 = 0;               /* for 12 */
  for (i = 1; i < 12; i = i+1)
    sum1 = sum1 + i*i;

  sum2 = 0;               /* for 24 */
  for (i = 1; i < 24; i = i+1)
    sum2 = sum2 + i*i;

  sum3 = 0;               /* for 30 */
  for (i = 1; i < 30; i = i+1)
    sum3 = sum3 + i*i;

  printf("%d, %d, %d\n",
    sum1, sum2, sum3);
  return 0;
}
```

link

Let's create a program, that will perform the previous tasks with numbers 12, 24 and 30!

Our solution

- was made by Copy+Paste+correct
- many possibilities for mistakes, errors
- long program

Segmentation – motivation

```
int main(void) {
  int i, sum1, sum2, sum3;

  sum1 = 0;           /* for 12 */
  for (i = 1; i < 12; i = i+1)
    sum1 = sum1 + i*i;

  sum2 = 0;           /* for 24 */
  for (i = 1; i < 24; i = i+1)
    sum2 = sum2 + i*i;

  sum3 = 0;           /* for 30 */
  for (i = 1; i < 30; i = i+1)
    sum3 = sum3 + i*i;

  printf("%d, %d, %d\n",
    sum1, sum2, sum3);
  return 0;
}
```

link

Let's create a program, that will perform the previous tasks with numbers 12, 24 and 30!
Our solution

- was made by Copy+Paste+correct
- many possibilities for mistakes, errors
- long program
- it is hard to manage

Segmentation – motivation

```c
int main(void) {
  int i, sum1, sum2, sum3;

  sum1 = 0;          /* for 12 */
  for (i = 1; i < 12; i = i+1)
    sum1 = sum1 + i*i;

  sum2 = 0;          /* for 24 */
  for (i = 1; i < 24; i = i+1)
    sum2 = sum2 + i*i;

  sum3 = 0;          /* for 30 */
  for (i = 1; i < 30; i = i+1)
    sum3 = sum3 + i*i;

  printf("%d, %d, %d\n",
    sum1, sum2, sum3);
  return 0;
}
```

link

Let's create a program, that will perform the previous tasks with numbers 12, 24 and 30!

Our solution

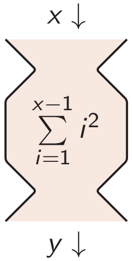
- was made by Copy+Paste+correct
- many possibilities for mistakes, errors
- long program
- it is hard to manage

Is it possible in a more smarter way?

Functions

The function

- Standalone program segment
- For operations that occur frequently
- We can run it (call it) with different arguments
- Calculates something, and gives back the result for the program that called it



Functions – solution

```c
int squaresum(int n) /* function definition */
{
  int i, sum = 0;
  for (i = 1; i < n; i = i+1)
    sum = sum + i*i;
  return sum;
}

int main(void) /* main program */
{
  int sum1, sum2, sum3;

  sum1 = squaresum(12); /* function call */
  sum2 = squaresum(24);
  sum3 = squaresum(30);

  printf("%d, %d, %d\n", sum1, sum2, sum3);
  return 0;
}
```

link

Function definition

Syntax of a function definition

```
<type of return value>
<function identifier> (<list of formal parameters>)
<block>
```

```c
int squaresum(int n)
{
  int i, sum = 0;
  for (i = 1; i < n; i = i+1)
    sum = sum + i*i;
  return sum;
}
```

Function definition

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of a function definition

```
<type of return value>
<function identifier> (<list of formal parameters>)
<block>
```

```
int squaresum(int n)
{
  int i, sum = 0;
  for (i = 1; i < n; i = i+1)
    sum = sum + i*i;
  return sum;
}
```

Function definition

Syntax of a function definition

```
<type of return value>
<function identifier> (<list of formal parameters>)
<block>
```

```c
int squaresum(int n)
{
  int i, sum = 0;
  for (i = 1; i < n; i = i+1)
    sum = sum + i*i;
  return sum;
}
```

Function definition

Syntax of a function definition

```
<type of return value>
<function identifier> (<list of formal parameters>)
<block>
```

```c
int squaresum(int n)
{
  int i, sum = 0;
  for (i = 1; i < n; i = i+1)
    sum = sum + i*i;
  return sum;
}
```

Function definition

Syntax of a function definition

<type of return value>
<function identifier> (<list of formal parameters>)
<block>

```c
int squaresum(int n)
{
  int i, sum = 0;
  for (i = 1; i < n; i = i+1)
    sum = sum + i*i;
  return sum;
}
```

Function definition

Syntax of a function definition

```
<type of return value>
<function identifier> (<list of formal parameters>)
<block>
```

```c
int squaresum(int n)
{
  int i, sum = 0;
  for (i = 1; i < n; i = i+1)
    sum = sum + i*i;
  return sum;
}
```

Function definition

Type of the return value:

- The type of the calculated value

```c
double average(int a, int b)
{
  return 0.5 * (a + b);
}
```

Function definition

Type of the return value:

- The type of the calculated value

```c
double average(int a, int b)
{
  return 0.5 * (a + b);
}
```

- or `void` (empty), if the function does not calculate anything

```c
void print_point(double x, double y)
{
  printf("(%.3f, %.3f)", x, y); /* (2.000, 4.123) */
}
```

Function definition

Type of the return value:

- The type of the calculated value

```c
double average(int a, int b)
{
  return 0.5 * (a + b);
}
```

- or void (empty), if the function does not calculate anything

```c
void print_point(double x, double y)
{
  printf("(%.3f, %.3f)", x, y); /* (2.000, 4.123) */
}
```

- because sometimes we don't care about the calculated value, only about the "side effect" (secondary effect).

A remark: Primary and secondary effects

Primary the function calculates and gives back the return value

A remark: Primary and secondary effects

Primary	the function calculates and gives back the return value
Secondary	the function ”performs some more things” (prints on screen, writes to file, plays an MP3, launches a missile. . .)

A remark: Primary and secondary effects

Primary	the function calculates and gives back the return value
Secondary	the function ”performs some more things” (prints on screen, writes to file, plays an MP3, launches a missile. . .)

- Some programming languages make a clear distinction between different program segments:

A remark: Primary and secondary effects

Primary	the function calculates and gives back the return value
Secondary	the function ”performs some more things” (prints on screen, writes to file, plays an MP3, launches a missile. . .)

- Some programming languages make a clear distinction between different program segments:

function	where the primary effect is the important

A remark: Primary and secondary effects

Primary	the function calculates and gives back the return value
Secondary	the function ”performs some more things” (prints on screen, writes to file, plays an MP3, launches a missile. . .)

- Some programming languages make a clear distinction between different program segments:

function	where the primary effect is the important
procedure	no primary effect, but the secondary effect is important

A remark: Primary and secondary effects

Primary	the function calculates and gives back the return value
Secondary	the function ”performs some more things” (prints on screen, writes to file, plays an MP3, launches a missile. . .)

- Some programming languages make a clear distinction between different program segments:

function	where the primary effect is the important
procedure	no primary effect, but the secondary effect is important

- In C language there is only function. Procedures are represented by functions with empty (`void`) return value.

A remark: Primary and secondary effects

Primary	the function calculates and gives back the return value
Secondary	the function ”performs some more things” (prints on screen, writes to file, plays an MP3, launches a missile. . .)

- Some programming languages make a clear distinction between different program segments:

function	where the primary effect is the important
procedure	no primary effect, but the secondary effect is important

- In C language there is only function. Procedures are represented by functions with empty (`void`) return value.
- Generally, we should try to separate the primary and secondary effects!

Function definition

Formal list of parameters

- Comma-separted list of declaration of parameters one-by-one, so we can reference them inside the function

```c
double volume(double x, double y, double z)
{
  return x*y*z;
}
```

Function definition

Formal list of parameters

- Comma-separted list of declaration of parameters one-by-one, so we can reference them inside the function

```
double volume(double x, double y, double z)
{
  return x*y*z;
}
```

- The number of parameters can be 0, 1, 2, . . . as much as you want (127 ☺)

Function definition

Formal list of parameters

- Comma-separted list of declaration of parameters one-by-one, so we can reference them inside the function

```c
double volume(double x, double y, double z)
{
  return x*y*z;
}
```

- The number of parameters can be 0, 1, 2, . . . as much as you want (127 ☺)
- If there are 0 parameters, we denote it with `void`

```c
double read_next_positive(void)
{
  double input;
  do scanf("%lf", &input) while (input <= 0);
  return input;
}
```

Function definition

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

The `return` statement

- it gives a return value, it terminates the execution of the function's block, and returns to the point of calling

Function definition

The `return` statement

- it gives a return value, it terminates the execution of the function's block, and returns to the point of calling
- there can be more of it, but it will cause to (terminate and) return to the point of calling at the first execution.

```c
double distance(double a, double b)
{
  double dist = b - a;
  if (dist < 0)
    return -dist;
  return dist;
}
```

Function definition

The `return` statement

- it gives a return value, it terminates the execution of the function's block, and returns to the point of calling
- there can be more of it, but it will cause to (terminate and) return to the point of calling at the first execution.

```c
double distance(double a, double b)
{
  double dist = b - a;
  if (dist < 0)
    return -dist;
  return dist;
}
```

- it can also occur in a `void`-type function `return;`

Function call

```c
double distance(double a, double b)
{
   ...
}
```

Syntax of a function call

`<function identifier> (<actual argument expr.>)`

```c
double x = distance(2.0, 3.0); /* x will be 1.0 */
```

Function call

```c
double distance(double a, double b)
{
   ...
}
```

Syntax of a function call

`<function identifier> (<actual argument expr.>)`

```c
double x = distance(2.0, 3.0); /* x will be 1.0 */
```

Function call

```c
double distance(double a, double b)
{
   ...
}
```

Syntax of a function call

`<function identifier> (<actual argument expr.>)`

```c
double x = distance(2.0, 3.0); /* x will be 1.0 */
```

Function call

```
double distance(double a, double b)
{
   ...
}
```

Syntax of a function call

<function identifier> (<actual argument expr.>)

```
double x = distance(2.0, 3.0); /* x will be 1.0 */
```

```
double a = 1.0;
double x = distance(2.5-1.0, a); /* x will be 0.5 */
```

Function call

```c
double distance(double a, double b)
{
   ...
}
```

Syntax of a function call

<function identifier> (<actual argument expr.>)

```c
double x = distance(2.0, 3.0); /* x will be 1.0 */
```

```c
double a = 1.0;
double x = distance(2.5-1.0, a); /* x will be 0.5 */
```

```c
double pos = read_next_positive(); /* empty () */
```

The main program as a function

```c
int main(void) /* now we understand, what this is */
{
   ...
   return 0;
}
```

The main program as a function

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```c
int main(void) /* now we understand, what this is */
{
  ...
  return 0;
}
```

The main program is also a function

- it is called by the operation system at the start of the program
- it does not get any arguments (we will change this later)
- it returns with integer (`int`) value
  - Traditionally, if execution was OK, it gives 0-t, otherwise an error code

```
Process returned 0 (0x0)   execution time:  0.001 s
press ENTER to continue.
```

Mechanism of function call

```
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;              /* base */
  b = 3;          /* height */
  T = area(a, b); /* area */
  return 0;
}
```

register: ??

Mechanism of function call

```
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;            /* base */
  b = 3;          /* height */
  T = area(a, b); /* area */
  return 0;
}
```

register: ??

The `main` function allocates space for its local variables in the stack.

Mechanism of function call

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;              /* base */
  b = 3;          /* height */
  T = area(a, b); /* area */
  return 0;
}
```

T 0x1FF8:	????
b 0x1FFC:	????
a 0x2000:	????

register:	??

The `main` function allocates space for its local variables in the stack.

Mechanism of function call

```
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;                 /* base */
  b = 3;             /* height */
  T = area(a, b); /* area */
  return 0;
}
```

T 0x1FF8:	????
b 0x1FFC:	????
a 0x2000:	????

register:	??

It gives values to its local variables.

Mechanism of function call

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;               /* base */
  b = 3;           /* height */
  T = area(a, b); /* area */
  return 0;
}
```

T	0x1FF8:	????
b	0x1FFC:	????
a	0x2000:	2

register:	??

It gives values to its local variables.

Mechanism of function call

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;             /* base */
  b = 3;         /* height */
  T = area(a, b); /* area */
  return 0;
}
```

T 0x1FF8:	????
b 0x1FFC:	????
a 0x2000:	2
register:	??

It gives values to its local variables.

Mechanism of function call

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;              /* base */
  b = 3;          /* height */
  T = area(a, b); /* area */
  return 0;
}
```

T 0x1FF8:	????
b 0x1FFC:	3
a 0x2000:	2

register:	??

It gives values to its local variables.

Mechanism of function call

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;              /* base */
  b = 3;          /* height */
  T = area(a, b); /* area */
  return 0;
}
```

T 0x1FF8:	????
b 0x1FFC:	3
a 0x2000:	2

register:	??

Function call: the main function creates a copy of variables b and a in the stack.

Mechanism of function call

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;            /* base */
  b = 3;          /* height */
  T = area(a, b); /* area */
  return 0;
}
```

	0x1FF4:	3
T	0x1FF8:	????
b	0x1FFC:	3
a	0x2000:	2

register:	??

Function call: the `main` function creates a copy of variables b and a in the stack.

Mechanism of function call

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;             /* base */
  b = 3;           /* height */
  T = area(a, b); /* area */
  return 0;
}
```

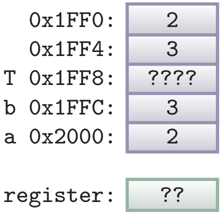
	0x1FF0:	2
	0x1FF4:	3
T	0x1FF8:	????
b	0x1FFC:	3
a	0x2000:	2

register:	??

Function call: the main function creates a copy of variables b and a in the stack.

Mechanism of function call

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;            /* base */
  b = 3;         /* height */
  T = area(a, b); /* area */
  return 0;
}
```



Function call: the main function places the return address in the stack.

Mechanism of function call

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;              /* base */
  b = 3;          /* height */
  T = area(a, b); /* area */
  return 0;
}
```

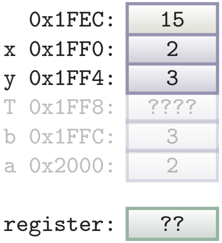
	0x1FEC:	15
	0x1FF0:	2
	0x1FF4:	3
T	0x1FF8:	????
b	0x1FFC:	3
a	0x2000:	2

register: ??

Function call: the main function places the return address in the stack.

Mechanism of function call

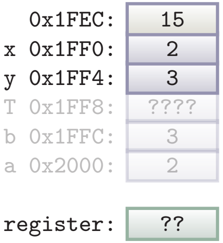
```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;            /* base */
  b = 3;          /* height */
  T = area(a, b); /* area */
  return 0;
}
```



The control is handed over to the area function, who will see the actual parameters as x and y

Mechanism of function call

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;              /* base */
  b = 3;           /* height */
  T = area(a, b); /* area */
  return 0;
}
```



The area function allocates space for variable S in the stack

Mechanism of function call

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;             /* base */
  b = 3;          /* height */
  T = area(a, b); /* area */
  return 0;
}
```

S	0x1FE8:	????
	0x1FEC:	15
x	0x1FF0:	2
y	0x1FF4:	3
T	0x1FF8:	????
b	0x1FFC:	3
a	0x2000:	2

register:	??

The area function allocates space for variable S in the stack

Mechanism of function call

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;            /* base */
  b = 3;          /* height */
  T = area(a, b); /* area */
  return 0;
}
```

S	0x1FE8:	????
	0x1FEC:	15
x	0x1FF0:	2
y	0x1FF4:	3
T	0x1FF8:	????
b	0x1FFC:	3
a	0x2000:	2

register: ??

It calculates the value of S

Mechanism of function call

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;            /* base */
  b = 3;          /* height */
  T = area(a, b); /* area */
  return 0;
}
```

S	0x1FE8:	6
	0x1FEC:	15
x	0x1FF0:	2
y	0x1FF4:	3
T	0x1FF8:	????
b	0x1FFC:	3
a	0x2000:	2

register:	??

It calculates the value of S

Mechanism of function call

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;              /* base */
  b = 3;          /* height */
  T = area(a, b); /* area */
  return 0;
}
```

S	0x1FE8:	6
	0x1FEC:	15
x	0x1FF0:	2
y	0x1FF4:	3
T	0x1FF8:	????
b	0x1FFC:	3
a	0x2000:	2

register: ??

It returns the value of S through a register.

Mechanism of function call

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;              /* base */
  b = 3;            /* height */
  T = area(a, b); /* area */
  return 0;
}
```

S	0x1FE8:	6
	0x1FEC:	15
x	0x1FF0:	2
y	0x1FF4:	3
T	0x1FF8:	????
b	0x1FFC:	3
a	0x2000:	2

register:	6

It returns the value of S through a register.

Mechanism of function call

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;            /* base */
  b = 3;          /* height */
  T = area(a, b); /* area */
  return 0;
}
```

S	0x1FE8:	6
	0x1FEC:	15
x	0x1FF0:	2
y	0x1FF4:	3
T	0x1FF8:	????
b	0x1FFC:	3
a	0x2000:	2

register:	6

It removes S from the stack.

Mechanism of function call

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;              /* base */
  b = 3;          /* height */
  T = area(a, b); /* area */
  return 0;
}
```

	0x1FEC:	15
x	0x1FF0:	2
y	0x1FF4:	3
T	0x1FF8:	????
b	0x1FFC:	3
a	0x2000:	2

register:	6

It removes S from the stack.

Mechanism of function call

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;          /* base */
  b = 3;        /* height */
  T = area(a, b); /* area */
  return 0;
}
```

	0x1FEC:	15
x	0x1FF0:	2
y	0x1FF4:	3
T	0x1FF8:	????
b	0x1FFC:	3
a	0x2000:	2

register:	6

The control is given back to calling program segment, to the line that was saved.

Mechanism of function call

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;           /* base */
  b = 3;         /* height */
  T = area(a, b); /* area */
  return 0;
}
```

	0x1FEC:	15
	0x1FF0:	2
	0x1FF4:	3
T	0x1FF8:	????
b	0x1FFC:	3
a	0x2000:	2

register:	6

The control is given back to calling program segment, to the line that was saved.

Mechanism of function call

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;            /* base */
  b = 3;          /* height */
  T = area(a, b); /* area */
  return 0;
}
```

```
  0x1FEC:   15
  0x1FF0:   2
  0x1FF4:   3
T 0x1FF8:  ????
b 0x1FFC:   3
a 0x2000:   2

register:   6
```

The `main` function copies the return value from the register.

Mechanism of function call

```
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;          /* base */
  b = 3;        /* height */
  T = area(a, b); /* area */
  return 0;
}
```

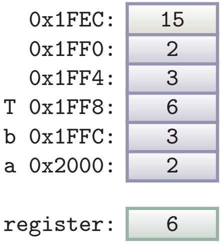
	0x1FEC:	15
	0x1FF0:	2
	0x1FF4:	3
T	0x1FF8:	6
b	0x1FFC:	3
a	0x2000:	2

register:	6

The main function copies the return value from the register.

Mechanism of function call

```
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;          /* base */
  b = 3;        /* height */
  T = area(a, b); /* area */
  return 0;
}
```

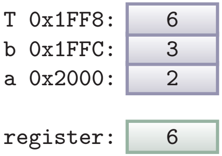


```
  0x1FEC: 15
  0x1FF0: 2
  0x1FF4: 3
T 0x1FF8: 6
b 0x1FFC: 3
a 0x2000: 2

register: 6
```

The main function removes the return address and the parameters from the stack.

Mechanism of function call

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;              /* base */
  b = 3;            /* height */
  T = area(a, b); /* area */
  return 0;
}
```



The function call is finished.

Mechanism of function call

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;              /* base */
  b = 3;          /* height */
  T = area(a, b); /* area */
  return 0;
}
```

T 0x1FF8:	6
b 0x1FFC:	3
a 0x2000:	2
register:	6

The main function copies return value 0 into the register.

Mechanism of function call

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;              /* base */
  b = 3;          /* height */
  T = area(a, b); /* area */
  return 0;
}
```

T 0x1FF8:	6
b 0x1FFC:	3
a 0x2000:	2

register:	0

The `main` function copies return value 0 into the register.

Mechanism of function call

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;              /* base */
  b = 3;            /* height */
  T = area(a, b); /* area */
  return 0;
}
```

T 0x1FF8:	6
b 0x1FFC:	3
a 0x2000:	2
register:	0

The `main` function removes its variables from the stack, and hands over the control to the operating system.

Mechanism of function call

```
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;          /* base */
  b = 3;        /* height */
  T = area(a, b); /* area */
  return 0;
}
```

register: 0

Mechanism of function call

Passing parameters by value

- Functions receive the value of the actual argument expressions.

Mechanism of function call

Passing parameters by value

- Functions receive the value of the actual argument expressions.
- Parameters can be used as variables, that have an initial value assigned at the point of calling.

Mechanism of function call

Passing parameters by value

- Functions receive the value of the actual argument expressions.
- Parameters can be used as variables, that have an initial value assigned at the point of calling.
- Functions may modify the values of the parameters, but this has no effect on the calling program segment.

Visibility and life-cycle of variables

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Local variables

1. parameters of functions

Global variables – only for emergency cases!

Variables declared outside of functions (even outside of `main()`)

Visibility and life-cycle of variables

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Local variables

1. parameters of functions
2. variables declared inside a function

Global variables – only for emergency cases!

Variables declared outside of functions (even outside of `main()`)

Visibility and life-cycle of variables

Local variables

1. parameters of functions
2. variables declared inside a function

- They are created when entering into the function, and are erased when returning from the function.

Global variables – only for emergency cases!

Variables declared outside of functions (even outside of `main()`)

Visibility and life-cycle of variables

Local variables

1. parameters of functions
2. variables declared inside a function

- They are created when entering into the function, and are erased when returning from the function.
- They are invisible for program segments outside of the function. (also for the calling segment!)

Global variables – only for emergency cases!

Variables declared outside of functions (even outside of `main()`)

Visibility and life-cycle of variables

Local variables

1. parameters of functions
2. variables declared inside a function

- They are created when entering into the function, and are erased when returning from the function.
- They are invisible for program segments outside of the function. (also for the calling segment!)

Global variables – only for emergency cases!

Variables declared outside of functions (even outside of `main()`)

- They exist throughout the life-cycle of the program.

Visibility and life-cycle of variables

Local variables

1. parameters of functions
2. variables declared inside a function

- They are created when entering into the function, and are erased when returning from the function.
- They are invisible for program segments outside of the function. (also for the calling segment!)

Global variables – only for emergency cases!

Variables declared outside of functions (even outside of `main()`)

- They exist throughout the life-cycle of the program.
- They are visible for everyone and can be modified by anyone!

Visibility and life-cycle of variables

Local variables

1. parameters of functions
2. variables declared inside a function

- They are created when entering into the function, and are erased when returning from the function.
- They are invisible for program segments outside of the function. (also for the calling segment!)

Global variables – only for emergency cases!

Variables declared outside of functions (even outside of `main()`)

- They exist throughout the life-cycle of the program.
- They are visible for everyone and can be modified by anyone!
- In case of conflicts, the local variable masks out the global one.

Riddle

What will the following program print on the screen?

```c
#include <stdio.h>

int a, b;

void func(int a)
{
  a = 2;
  b = 3;
}

int main(void)
{
  a = 1;
  func(a);
  printf("a: %d, b: %d\n", a, b);
  return 0;
}
```

link

A complex task

Let's create a C program, that asks two integer numbers from the user (`low` $<$ `high`), and lists all prime numbers between these two numbers..

A complex task

Let's create a C program, that asks two integer numbers from the user (`low` < `high`), and lists all prime numbers between these two numbers..

- Pseudo-code of the solution broken into segments:

mainprogram

```
IN: low, high
FOR EACH i
  between low and high
  IF primetest(i) TRUE
    OUT: i
```

primetest(p)

```
FOR EACH i
  between 2 and root of p
  IF i divides p
    return FALSE
return TRUE
```

A complex task

Let's create a C program, that asks two integer numbers from the user (`low` $<$ `high`), and lists all prime numbers between these two numbers..

- Pseudo-code of the solution broken into segments:

mainprogram

```
IN: low, high
FOR EACH i
  between low and high
  IF primetest(i) TRUE
    OUT: i
```

primetest(p)

```
FOR EACH i
  between 2 and root of p
  IF i divides p
    return FALSE
return TRUE
```

- Notice the role of the two *i* and *p*

Complex task – solution

```
#include <stdio.h> /* scanf, printf */

int low, high; /* global variables */

void read(void) /* inputting function */
{
  printf("Give a small and a larger number!\n");
  scanf("%d%d", &low, &high);
}

int isprime(int p) /* primetest function. */
{
  int i;
  for (i=2; i*i<=p; i=i+1) /* i from 2 to root of p */
    /* if p is dividable by i, not a prime */
    if (p%i == 0)
      return 0;
  return 1; /* if we get here, it is a prime */
}
```

Complex task – solution

```c

int main()
{
  int i;

  read(); /* we read the limits with a function */

  printf("Primes between %d and %d:\n", low, high);
  for (i=low; i<=high; i=i+1)
  {
    if (isprime(i)) /* we test with a function */
      printf("%d\n", i);
  }

  return 0;
}
```

link

Design principles

- Functions and programs communicate via parameters and return values.
- Except when this is their special task, functions
    - do not print on the screen,
    - do not read from keyboard,
    - do not use global variables.

Thank you for your attention.