EECS2031 - Software Tools

Functions and Program Structure
(K&R Ch.1.5-10, Ch.4)

YORK UNIVERSITÉ UNIVERSITY

1

- C program structure – Functions
  - Communication
  - “Pass-by-value”
- Categories, scope and life time of variables (and functions)
- C Preprocessing
- Recursions

YORK UNIVERSITÉ UNIVERSITY

2

Program Structure

- C programs consist of a set of variables and functions.
  - we have discussed variables, expressions (ch2) and control flow (ch3).
  - now lets combine these into a program (ch4)
- C programs consist of statements
  - expression statements (ch2)
  - control flow statements (ch3)
  - block, function call statements (ch4)

Declaring Functions (review)

- Either a declaration or a definition must be present prior to any call of the function.

- Declaring a function before using it, if it is defined in
  - library e.g., include <stdio.h>
  - later in the same source file
  - another source file of the program
- Declaring a function tells its **return type** and **parameters** but not its code.

```
int power (int base, int pow);
```

- We can omit parameter names

```
int power  (int, int);
```

  - The type of parameters is what matters for compiler

Program structure -- Functions

- A function is a set of statements that may have:
  - a number of parameters --- values that can be passed to it
  - a return type that describes the value of this function in an expression

```
int sum (int a, int b)
{
      ….
}

int x,y
int a = sum(x, y)
```

"parameters", "formal parameters"

"arguments", "actual parameters"

Program structure -- Functions

- A function is a set of statements that may have:
  - a number of parameters --- values that can be passed to it
  - a return type that describes the value of this function in an expression
- Communication between functions
  - by arguments and return values
  - by external variable (ch1.10, ch4.3)
- Functions can occur
  - In a single source file
  - In multiple source files

Program structure -- functions
communication by arguments and return values

```
return_type  functionName (parameter type name, ……)
{block}
```

```
int sum (int i, int j){
  int s = i + j;
  return s;
}
```

```
void display (int i){
  printf("this is %d", i);
}
```

```
int main(){
  int x =2, y=3;
  int su = sum(x,y);
  display(su);  /* this is 5 */
  display( sum(x,y) );
}
```

Same in Java

7

- Communication by arguments and return values

Program structure -- functions
communication by external variables

```
#include <stdio.h>

int resu;       /* external/global variable  */
                /* defined outside any function */
void sum (int i, int j){
   resu = i + j;
}

int main(){
  int x =2, y =3;
  sum(x,y);
  printf("%d + %d = %d\n", x,y, resu);
}
```

8

Functions
communication by external variables

another example

```
#include <stdio.h>

  int resu;              /*  external/global variable  */

 void sum (int i, int j){
    resu = i + j;  /* grab resu */
  }

 display(){
    printf("this is %d\n", resu); /* grab resu */
  }

 int main(){
   int x =2, y =3;
   sum(x,y);
   display();  /* this is 5  */
 }
```

Easier communication

9

9

Functions
communication by external variables

one more example

```
#include <stdio.h>

  int resu;              /*  external variable  */

 void increase (){
    resu += 100;  /* grab resu */
  }

 void decrease(){
    resu -= 30;   /* grab resu */
  }

 int main(){
   resu=0;
   increase();
   decrease();
   printf("%d", resu);  // ?
 }
```

Easier communication

10

10

Multiple source files

C program with two source files

f1.c

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

'extern' can be omitted (for function)

f2.c

```
#include <stdio.h>

extern int sum(int, int);
                // declare
int main(){
 int x =2, y =3;
 printf("%d + %d = %d\n",
            x,y,sum(x,y));
}
```

To compile: `gcc f1.c f2.c`
`gcc f2.c f1.c`

11

YORK UNIVERSITÉ UNIVERSITY

Multiple source files

C program with two source files

f1.c

```
//define global variable
int resu;

// define functions
int sum (int x, int y)
{
   resu = x + y;
}
```

f2.c

```
#include <stdio.h>
extern int sum(int, int);
extern int resu; // declare

int main(){
 int x =2, y =3;
 sum(x,y);
 printf("%d\n", resu);
}
```

To compile: `gcc f1.c f2.c`
`gcc f2.c f1.c`

12

YORK UNIVERSITÉ UNIVERSITY

Declaring external variables

- Declaring a function before using it, if it is defined in
  - library e.g., `include <stdio.h>  extern int printf(….)`
  - later in the same source file
  - another source file of the program

- Declaring a global variable before using it, if it is defined in
  - library
  - later in the same source file
  - another source file of the program

	Definition the compiler allocates memory for that variable/function	**Declaration** informs the compiler that a variable/function by that name and type exists, so does not need to allocate memory for it since it is allocated elsewhere.

Function	`int sum (int j, int k){`
	`  return j+k;`
	`}`
		`int sum(int, int);`
		or
		`extern int sum(int, int);`

variable	`int i;`	`extern int i;`

13

- C program structure – Functions
  - Communication
  - “Pass-by-value”
- Categories, scope and life time of variables (and functions)
- C Preprocessing
- Recursion

14

YORK UNIVERSITÉ UNIVERSITY

14

Call (pass)-by-Value vs by reference

- So what is the question?

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

main(){
  int i=3, j=4, k;
  k = sum(i,j);
}
```

When sum() is called with sum(i,j), what happens to arguments i and j?

sum() got i, j themselves

or,

sum() got copies of i, j?

Call (pass)-by-Value vs by-reference

- So what is the question?

When `sum(int x, int y)` is called with `sum(i,j)`, what happens to arguments i, j?

- i and j themselves passed to sum -- "**pass by reference**"
  - x, y are alias of i,j x++ change i
- copies of i, j are passed to sum -- "**pass by value**"
  - x, y are copies of i,j x++ does not change i

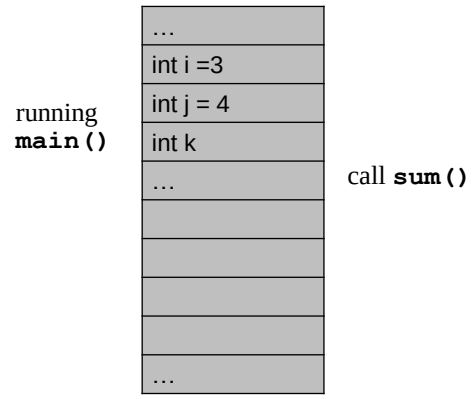
Difference between call by value and call by reference

No.	Call by value	Call by reference
1	A copy of value is passed to the function	An address of value is passed to the function
2	Changes made inside the function is not reflected on other functions	Changes made inside the function is reflected outside the function also

Call (pass)-by-Value

- In C, all functions are called-by-value
  - Value of the arguments are passed to functions, but not the arguments themselves (call-by-reference)

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

main(){
  int i=3, j=4, k;
  k = sum(i,j);
}
```

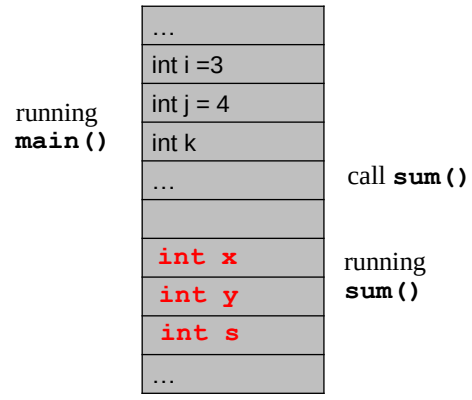


17

Call (pass)-by-Value

- In C, all functions are called-by-value
  - Value of the arguments are passed to functions, but not the arguments themselves (call-by-reference)

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

main(){
  int i=3, j=4, k;
  k = sum(i,j);
}
```

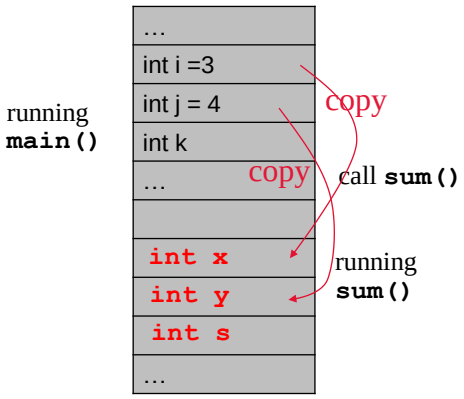


18

Call (pass)-by-Value

- In C, all functions are called-by-value
  - Value of the arguments are passed to functions, but not the arguments themselves (call-by-reference)

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

main(){
  int i=3, j=4, k;
  k = sum(i,j);
}
```

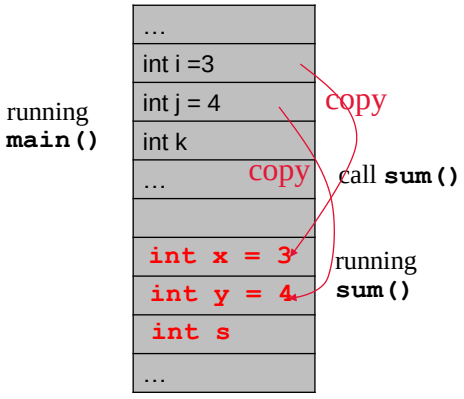


19

Call (pass)-by-Value

- In C, all functions are called-by-value
  - Value of the arguments are passed to functions, but not the arguments themselves (call-by-reference)

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

main(){
  int i=3, j=4, k;
  k = sum(i,j);
}
```



20

- The fact that arguments are passed by value has both advantages and disadvantages.
- Since a parameter can be modified without affecting the corresponding (actual) argument, we can use parameters as (local) variables within the function, reducing the number of genuine variables needed

```
int p = 5;  power(10,p);
```

Disadvantages? →

```
int power(int x, int n)
{
   int i, result = 1;

   for (i = 1; i <= n; i++)
     result = result * x;

   return result;
}
```

Since `n` is a *copy* of the original exponent p, the function can safely modify it, removing the need for `i`: →

```
int power(int x, int n)
{
   int result = 1;

   while (n > 0){
     result = result * x;
     n--; // p not affected
   }
   return result;
}
```

21

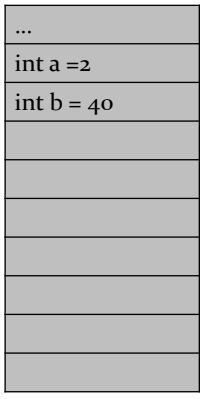
For your information

Call-by-Value does this code work?

```
void increment(int x, int y)
{

   x ++;
   y += 10;
}

void main( ) {
   int a=2, b=40;

   increment( a, b);
   printf("%d %d", a, b);
}
```

22

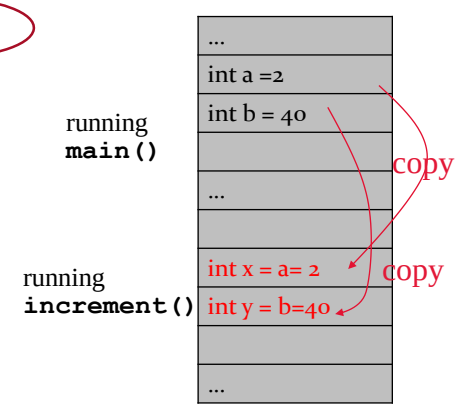
running `main()`



Call-by-Value
does this code work?

```
void increment(int x, int y)
{
    x ++;
    y += 10;
}

void main( ) {
    int a=2, b=40;

    increment( a, b);
    printf("%d %d", a, b);
}
```

Pass by value !!!



23

23

Call-by-Value
does this code work?

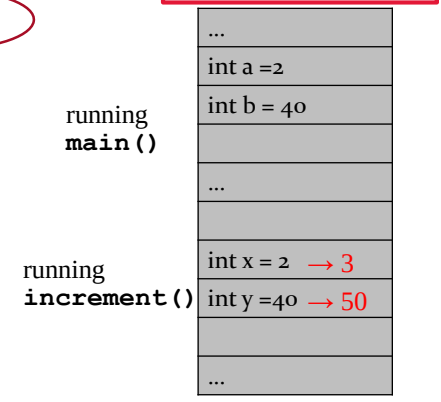
```
void increment(int x, int y)
{
    x ++;
    y += 10;
}

void main( ) {
    int a=2, b=40;

    increment( a, b);
    printf("%d %d", a, b);
}
```

Pass by value !!!

2 40

same in Java

a b not incremented !

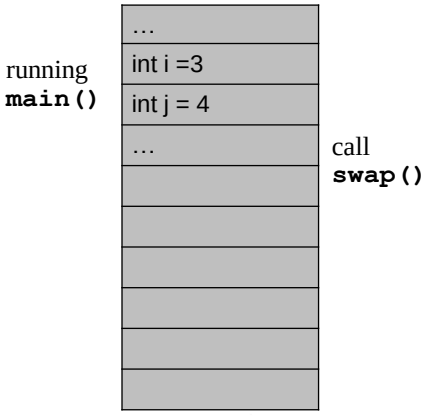


24

24

Call-by-Value
does this code work?

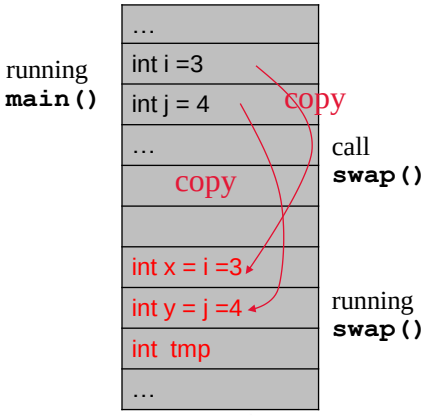
```
#include <stdio.h>

void swap (int x, int y)
{ int temp;
  temp = x;
  x = y;
  y = temp;
}

int main(){
  int i=3, j=4;
  swap(i,j);
  printf("%d %d\n", i,j);
}
```



25

Call-by-Value
does this code work?

```
#include <stdio.h>

void swap (int x, int y)
{ int temp;
  temp = x;
  x = y;
  y = temp;
}

int main(){
  int i=3, j=4;
  swap(i,j);
  printf("%d %d\n", i,j);
}
```



26

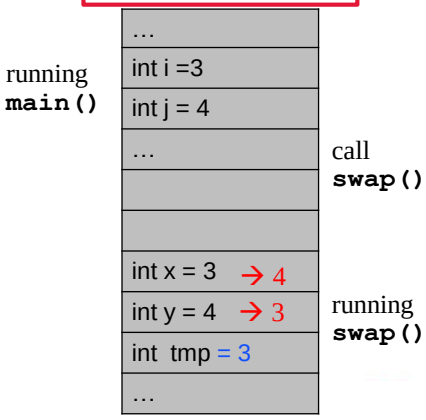
Call-by-Value
does this code work?

```
#include <stdio.h>

void swap (int x, int y)
{ int temp;
  temp = x;
  x = y;
  y = temp;
}

int main(){
  int i=3, j=4;
  swap(i,j);
  printf("%d %d\n", i,j);
}
```

3 4

same in Java

i j not affected !



Mr. Main

Hi, function, I have some manuscripts, stored in lockers (memory), and I want you to repaint them so their color changed.

Ms function

Hi, Mr Main, here is how we function work:
We don't take your original manuscript over the counter (not pass by reference). We always photocopy things passed here (pass by value) , and work on copies.

Mr. Main:
Then, is there a way to have my original manuscripts' color changed by you?

Mr. Main

Ms function:
Don't worry. Of course there is a way. Give us your locker address (pointer) instead ….

Summary and future work

- C program structure – Functions
  - Communication. Extern/global variables
  - Pass-by-value
- Categories, scope and life time of variables (and functions)
- C Preprocessing
- Recursions
- Other C materials before pointer
  - 2D array, string manipulations
  - Common library functions [Appendix of K+R]
- Pointers !

Next time

In class exercise – 1

write down your name, student # -- your identify is more critical than correctness!

1. You want an integer whose binary representation is

   0000 ……. 01011101

   What are the three ways to write it in C? (Don't use arithmetic operators)

   int a =

   int a =

   int a =

2. Given int i = 025;

   What is the decimal and binary representation of i?

In class exercise – 1

write down your name, student # -- your identify is more critical than correctness!

1. You want an integer whose binary representation is

   0000 ……. 01011101

   What are the three ways to write it in C? (Don't use arithmetic operators)

   int a = 93
   int a = 0135
   int a = 0x5D

2. Given int i = 025;

   What is the decimal and binary representation of i?

31 21 000….00010101