1

- C program structure – Functions (ch4)
  - Communication
  - "Pass-by-value"
- Categories, scope and lifetime of variables (ch4)
- C Preprocessing (ch4)

Last two lectures

- Recursions (ch4)
- Other C material before pointer
  - C library functions
  - 2D array

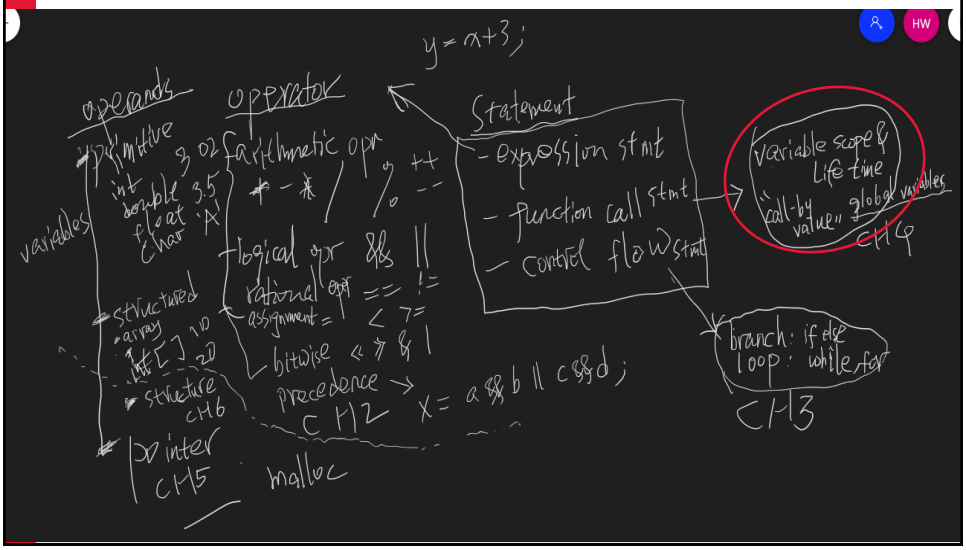
6

YORK UNIVERSITÉ UNIVERSITY

6

Roadmap -- How the topics are related

RECALL



7

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

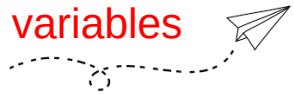
8

YORK UNIVERSITÉ UNIVERSITY

8

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 = 50;
     increase();
     decrease();
     printf("%d", resu);  // ?  120
   }
```

Easier communication

9

9

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

10

YORK UNIVERSITÉ UNIVERSITY

10

Multiple source files

Can call a function defined in another file. How 

functions.c

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

main.c

```
#include <stdio.h>


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

C program with two source files

Multiple source files

Can call a function defined in another file. How 

functions.c

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

main.c

```
#include <stdio.h>
#include "functions.c"

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

Works, but not a good practice

```
gcc main.c
```

https://stackoverflow.com/questions/31002266/why-we-should-not-include-source-files-in-c/31002641

Multiple source files

Declaring a function before using it, if defined in
- library e.g., include <stdio.h>
- later in the same source file
- another source file of the program

functions.c

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

'extern' can be omitted (for function)

main.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 main.c functions.c`
`gcc functions.c main.c`

13

Multiple source files

Declaring a function or global variable before using it, if defined in
- library e.g., include <stdio.h>
- later in the same source file
- another source file of the program

Can use a global variable defined in another file.

How ? Declare it!

'extern' can be omitted (for function)

functions.c

```
//define global variable
int resu;

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

main.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 main.c functions.c`
`gcc functions.c main.c`

14

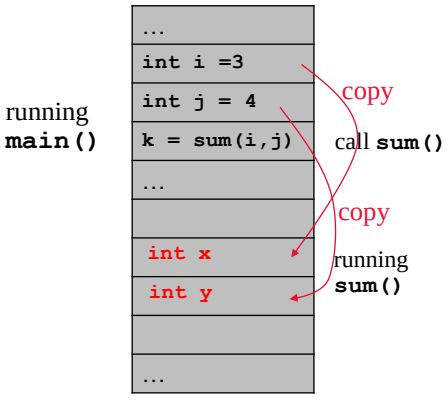
- C program structure – Functions
  - Communication
  - Pass-by-value
- Categories, scope, lifetime and initialization of variables (and functions)
- C Preprocessing
- Recursions

YORK UNIVERSITÉ UNIVERSITY

Call (pass)-by-Value

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

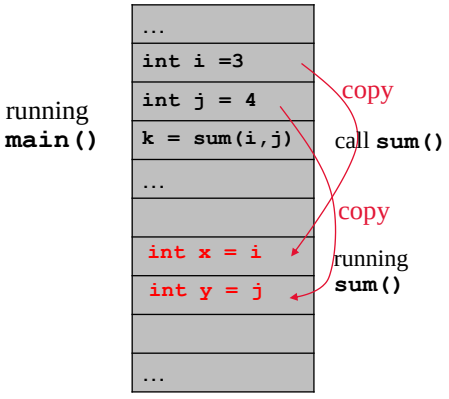
```
int sum (int x, int y)
{
   int s = x + y;
   return s;
}

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

Call (pass)-by-Value

- In C (and JAVA), all functions are call-by-value
  - Values 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);
}
```

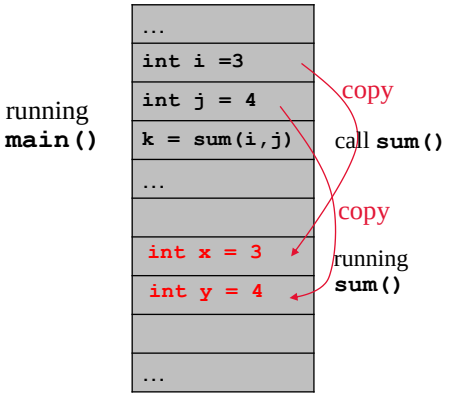


17

Call (pass)-by-Value

- In C (and JAVA), all functions are call-by-value
  - Values 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);
}
```

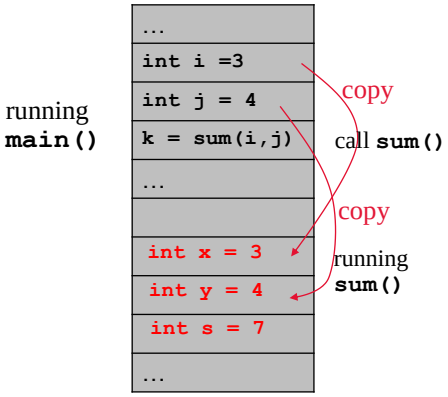


18

Call (pass)-by-Value

- In C (and JAVA), all functions are call-by-value
  - Values 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);
}
```

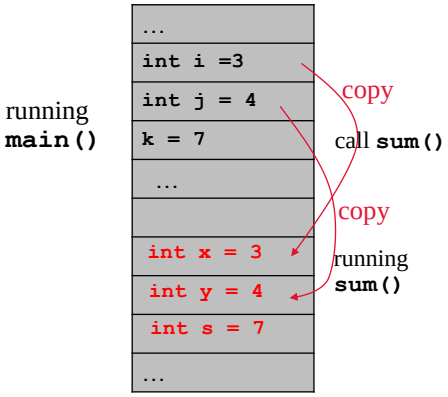


19

Call (pass)-by-Value

- In C (and JAVA), all functions are call-by-value
  - Values 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

- Write a short program to determine if a language is pass-by-value or pass-by-reference
- Simple

```
void increment(int x)
{
   x = x + 1;
}
void main( ) {
   int a=1;
   increment(a);
   print a;
}
```

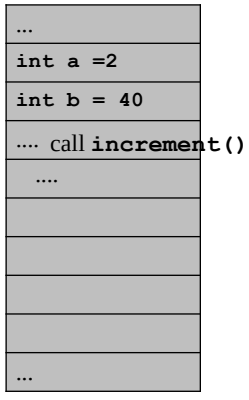
Output 1: pass by value
Output 2: pass by reference

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);
}
```


running **main()**



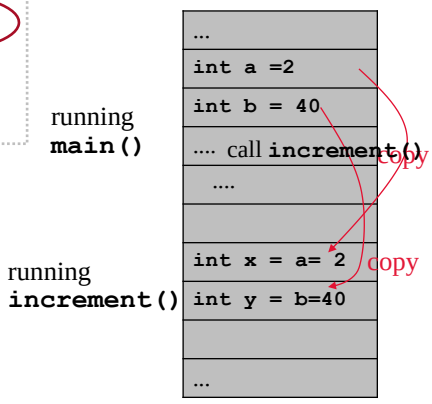
Call-by-Value
does this code work?

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

}
```

Pass by value !!!

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

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

26



26

Call-by-Value
does this code work?

```
void increment(int x, int y)
{
    x ++;
    y += 10;
    printf("%d %d", x, y);
}
```

Pass by value !!!

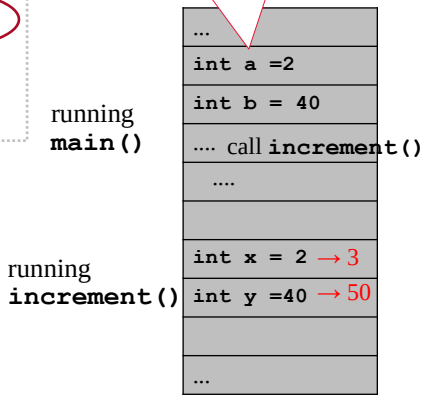
```
void main( ) {
    int a=2, b=40;

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

27

2 40

same in Java

a b not incremented !



27

- C program structure – Functions
  - Communication
  - Pass-by-value
- Categories, scope, lifetime and initialization of variables (and functions)
- C Preprocessing
- Recursions

Scope

- Scope of a name (variable or function) – the part of program within which the name can be used – spatial property
- Global variable (and functions) are all global! Outside any (other) function
- Automatic (local) variables: only exist within their blocks (main, loop...):

```
.....
{
  int x;
  .....
  {
    int y; /* y defined here */
    .....
  }
  ..... /* y not accessible here */
}
x++; /* x not accessible here */
```

same in Java

```
for(int i=0; i< 10;i++){
    int c = i+10;
}
System.out.println(i);
System.out.println(c);
```

c cannot be resolved to a variable
i cannot be resolved to a variable

i c defined in for loop.
not accessible after loop

```
error: 'x' undeclared (first use in this function)
```

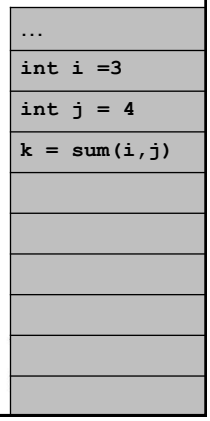
Lifetime (storage duration) -- temporal property
automatic (local) variables

- Come to life (allocated) the moment the function it is in is invoked/activated.
- Vanishes (deallocated) when the enclosing function returns!!!
- Values are not retained between function calls.

```
int sum (int x, int y)
{
  int s = x + y;
  return s;
}
main(){
  int i=3, j=4, k;
  k = sum(i,j);
  printf ("Sum is %d",k);
}
```

30

call **sum()**



30

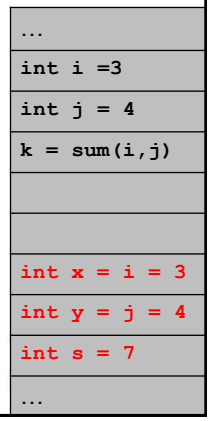
Lifetime – (storage duration)
automatic (local) variables

- Come to life (allocated) the moment the function it is in is invoked/activated,
- Vanishes (deallocated) when the enclosing function returns!!!
- Values are not retained between function calls.

```
int sum (int x, int y)
{
  int s = x + y;
  return s;
}
main(){
  int i=3, j=4, k;
  k = sum(i,j);
  printf ("Sum is %d",k);
}
```

31

vanish after sum() returns



31

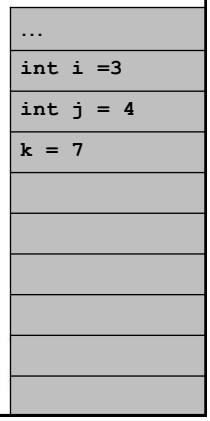
Lifetime – (storage duration)
automatic (local) variables

- Come to life (allocated) the moment the function it is in is invoked/activated,
- Vanishes (deallocated) when the enclosing function returns!!!
- Values are not retained between function calls.

```
int sum (int x, int y)
{
  int s = x + y;
  return s;
}
main(){
  int i=3, j=4, k;
  k = sum(i,j);
  printf ("Sum is %d",k);
}
```

vanish after sum() returns

i j?



32

32

Lifetime – (storage duration)
automatic (local) variables

```
void unique_int(void) {
  int counter = 0;
  printf("%d", counter);
  counter++;
}
main(){
  unique_int();
  ......
  unique_int();
  unique_int();
```

- The value of local variable **`counter`** is not preserved between calls to "**`unique_int()`**"
- By end of function, **`counter`** is 1, but then vanishes.
- Every function call creates a brand new **`counter`**

33

33

Lifetime
external (global) variables

- Permanent, as long as the program stays in memory
  - Retain values from one function to the next
- Can be used as an alternative for communication data between functions

```
int counter = 0;
void unique_int(void) {
   printf("%d", counter);
    counter++;
}
main(){
  unique_int();
  unique_int();
  ……
  unique_int();
```

34

- But use it with caution!

static declaration

static keyword have different meanings

- For a global variable or function,
  - hide it from other files. Limit the **scope** to the rest of the source file (only)

```
static int resu;
```

- For a local variable,
  - make its **lifetime** persistent

```
function(){
  static int i; // will not vanish
}
```

36

static (Hiding global variable)

calc.c

```
int x;
static int y;

void func1 (void)
{
   x--;
   y++; /* y still be
           accessed (later) in
           this file */
}
```

main.c

```
#include <stdio.h>

extern void func1(void);
extern int x;
extern int y;

int main(){
 x = 5; y = 10;
 func1()
 printf("%d %d\n", x,y);
}
```

37

What are outputs? *Does not compile* -- *" undefined reference to 'y' "*

37

static (Persistent local variables)

```
void unique_int(void) {
   static int counter = 0;
   printf("%d", counter);
   counter++;
}
main()
 unique_int();
 …
 unique_int();
 unique_int();
```

`printf("%d",counter);` 

- The value of local variable **`counter`** is preserved between calls to "**`unique_int()`**". **`counter`** is not dead!

```
int unique_int(void) {
   static int counter;
   printf("%d", counter);
   counter++;
}
```

39

- Initial value of **`counter`**

39

Initialization of variables

- For global (static or not) variable and static local variable
  - Initialization takes place at the compiling time before program is invoked
  - Initialized to 0 for int if no explicit initial value is given
    - So first call to **`unique_int()`** returns 0 even **`counter`** not initialized

```
int resu;
void decrease(){
   resu -= 30;
 }
int main(){
    decrease();
    printf("%d", resu);
}
```

global

```
int unique_int(void) {
   static int counter;
  printf("%d", counter);
  counter++;
}
unique_int();
unique_int();
unique_int();
```

static local

40

40

Initialization of variables

- For regular (non-static) local variables
  - If no explicit initial value, initial values are undefined (not initialized for you). May get garbage value.

```
int counter;    /* counter could be 45873972 */
while ( (c = getchar()) != EOF){
  counter++;
}
```

Compiles, but weird results

```
arr[20];
int index;    /* index could be 873972 */
while (index < 20){
   arr[index]=0;
   index++;
}
```

Java also doesn't initialize local variables, but let you know. 'variable index might not have been initialized'

41

41

Initialization of variables

- For regular (non-static) local variables
  - If no explicit initial value, initial values are undefined (not initialized for you). May get garbage value.

```
int occurrence(char arr[], char c){
  int count; int i;
  for(i=0; arr[i]!= '\0'; i++)
     if(arr[i] == c)
       count++;
  return count;
```

Compiles, but weird results

```
int length(char arr[]){
    int i;          /* i could be 1873972 */
    while (arr[i] != '\0')
      i++;
    return i;
}
```

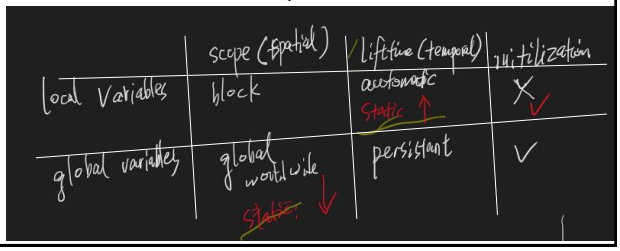
Java also doesn't initialize local variables, but let you know. 'variable index might not have been initialized'

42

42

Summary of Categories, scope, life time and initialization of variables

- Four different categories
  - External (global) variable
  - static global variable
  - Local (automatic, internal) variable
  - static local variable
- What are the difference between them, in terms of
  - scope
  - lifetime
  - initialization



43

43

- C program structure – Functions (ch4)
  - Communication
  - "Pass-by-value"
- Categories, scope and lifetime of variables (ch4)
- C Preprocessing (ch4)
- Recursions (ch4)
- Other C material before pointer
  - C library functions
  - 2D array

Last lecture

How C Programs are Compiled

- C programs go through three stages to be compiled:
  - Preprocessor - handles #include and #define etc
  - Compiler - converts C code into binary processor instructions ("object code")
  - Linker - puts multiple files together, load library function (e.g. printf, strlen) and creates an executable program

#define -- parameterized

- Macros can also have arguments

e.g.

```
#define TRIPLE(x)  3 * x

y = TRIPLE(4);
```

becomes

```
y = 3 * 4;
```

```
#define SQUARE(x)  x*x

y = SQUARE(5);
```

becomes

```
y = 5*5;
```

---

e.g., `#define MY_PRINT(x,y) printf("%d %d\n", x,y)`

```
MY_PRINT(3,5);
```

becomes

```
printf("%d %d\n", 3,5);
```

57

#define – Be careful with operators

```
#define TWO_PI  2*3.14

    double overpi = 1/ TWO_PI;
```

becomes

```
    double overpi = 1/2*3.14;        // 0
```

Fix: Use parentheses defensively, e.g.

`1.0/ TWO_PI;?`

```
#define TWO_PI (2*3.14)

    double overpi = 1/ TWO_PI;
```

becomes

```
    double overpi = 1/(2*3.14);      // 0.123..
```

Rule1: if replacement list contains operator, use () around whole replacement list

58

#define – parameterized. Be careful with arguments

```
#define TRIPLE(x)   3 * x

     y = TRIPLE(5+2);
```

```
#define SQUARE(x)   x*x

     y = SQUARE(5+2);
```

60

#define – parameterized. Be careful with arguments

```
#define TRIPLE(x)   3 * x

     y = TRIPLE(5+2);
```

becomes

```
     y = 3 * 5+2;      // 17
```

Fix: Use parentheses defensively, e.g.

```
#define TRIPLE(x)    3 * (x)

     y = TRIPLE(5+2);
```

becomes

```
     y = ((5+2) * 3);       // 21
```

1/TRIPLE(3.4+2)

1.0/TRIPLE(3+2)

500/TRIPLE(3+2)

Rule2: for parameterized, put () around each parameter occurrence in the replacement list

61

#define – parameterized. Be careful with arguments

```
#define SQUARE(x)  x*x

      y = SQUARE(5+2);
```

becomes

```
      y = 5+2*5+2;          // 17
```

Fix: Use parentheses defensively, e.g.

```
#define SQUARE(x)  ((x)*(x))

      y = SQUARE(5+2);
```

becomes

```
      y = ((5+2)*(5+2));    // 49
```

1/SQUARE(3.4+2)

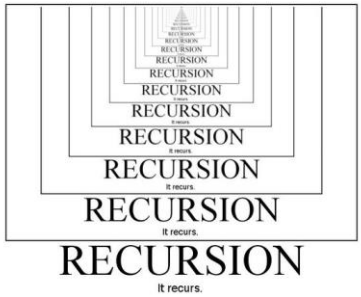
1.0/SQUARE(3+2)

500/SQUARE(3+2)

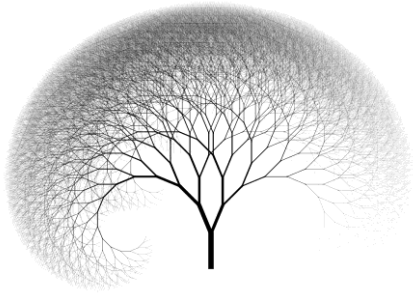
Rule2: for parameterized, put () around each parameter occurrence in the replacement list

- C program structure – Functions (ch4)
  - Communication
  - "Pass-by-value"
- Categories, scope and lifetime of variables (ch4)
- C Preprocessing (ch4)
- Recursions (ch4)
- Other C material before pointer
  - C library functions
  - 2D array

Last lecture



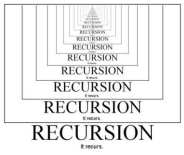
Think recursively



67

67

Think recursively



"how many people are there, from you to the end of line?"

" 1 + the answer of the guy behind me"

68

68

Recursive Algorithm: smaller instance

- Assume you have an algorithm that works.
- Use it to write an algorithm that works.

To get into my house
I must get the key from a smaller house
Assume I made it.

69

Recursion

"To get into my house
I must get the key from a smaller house

```
getIn (house A)
{
      getIn (smaller house on the right)
      pick up key for house A
      open A;
}
```

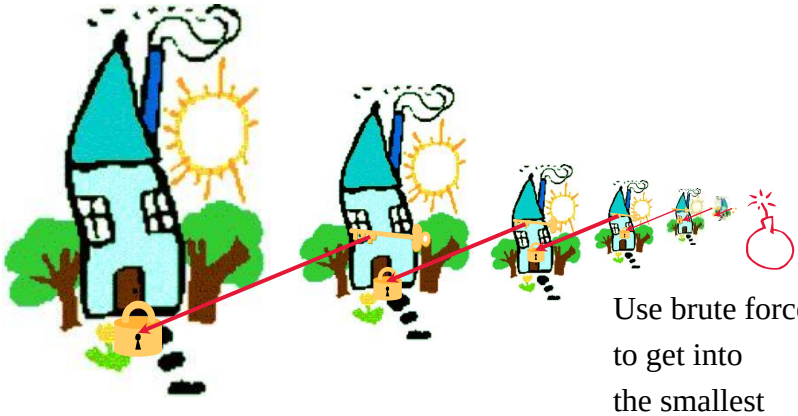
70

YORK UNIVERSITÉ UNIVERSITY

70

Recursive Algorithm: base cases

- Assume you have an algorithm that works.
- Use it to write an algorithm that works.



Use brute force to get into the smallest (shabbiest) house.

71

Recursion

"To get into my house
I must get the key from a smaller house

```
getIn (house A)
{
   if (A is the last/smallest)
     get in using brute force;

   else
     getIn (house on the right)
     pick up key for A
     open A;
}
```

YORK UNIVERSITÉ UNIVERSITY

72

Recursion

"To get into my house
I must get the key from a smaller house

```
int length (string s)
{   if (s contains no letter)
      return 0;
    return 1 + length(substring on the right);
}
```

length("ABCD")
= 1 + length("BCD")
= 1 + (1 + length("CD"))
= 1 + (1 + (1 + length("D")))
= 1 + (1 + (1 + (1+ length(""))))
= 1 + (1 + (1 + (1+ 0))) = 4

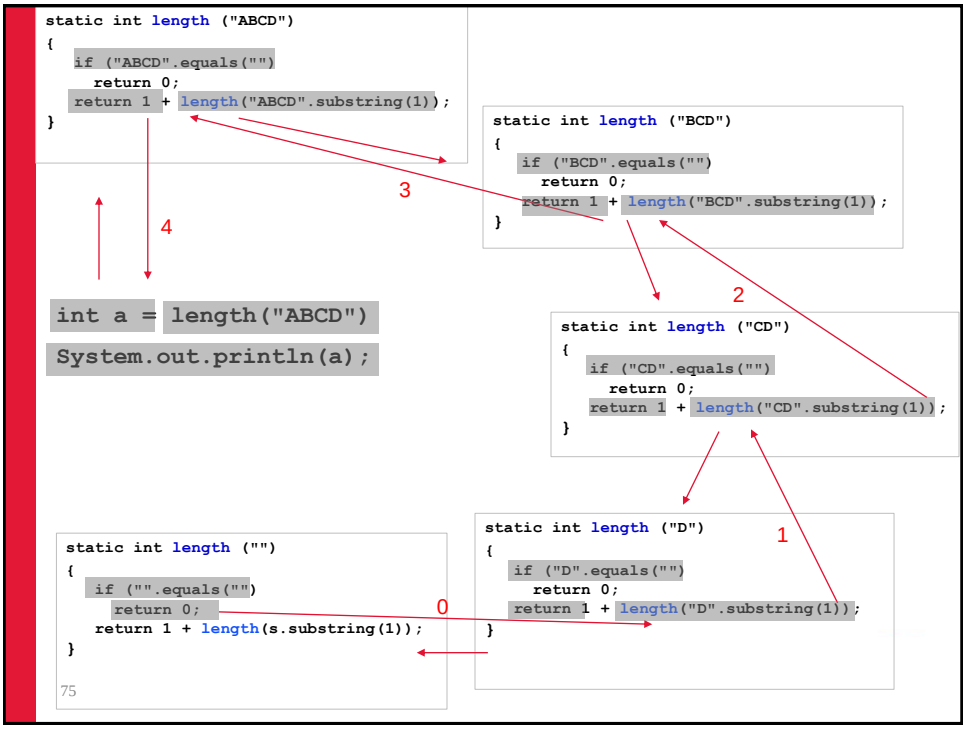
Recursion

"To get into my house
I must get the key from a smaller house

```
static int length (String s)
{   if (s.equals("") // contains no letter
      return 0;
    return 1 + length(s.substring(1));
}
```

length("ABCD")
= 1 + length("BCD")
= 1 + (1 + length("CD"))
= 1 + (1 + (1 + length("D")))
= 1 + (1 + (1 + (1+ length(""))))
= 1 + (1 + (1 + (1+ 0))) = 4

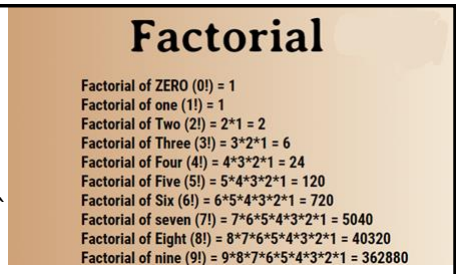
C version 



75

Recursion

- C supports recursion
- Think/define recursively



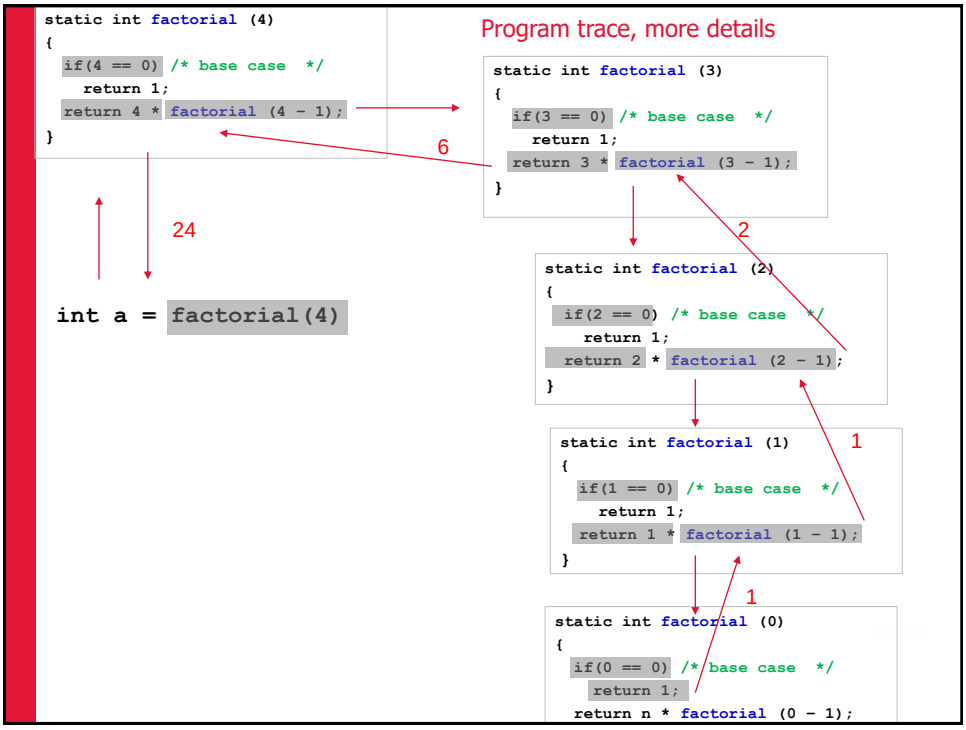
$$factorial(n) = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot factorial(n-1) & otherwise \end{cases}$$

```
int factorial (int n)
{
  if(n == 0) /* base case  */
   return 1;
  else
   return n * factorial (n - 1);
}
```

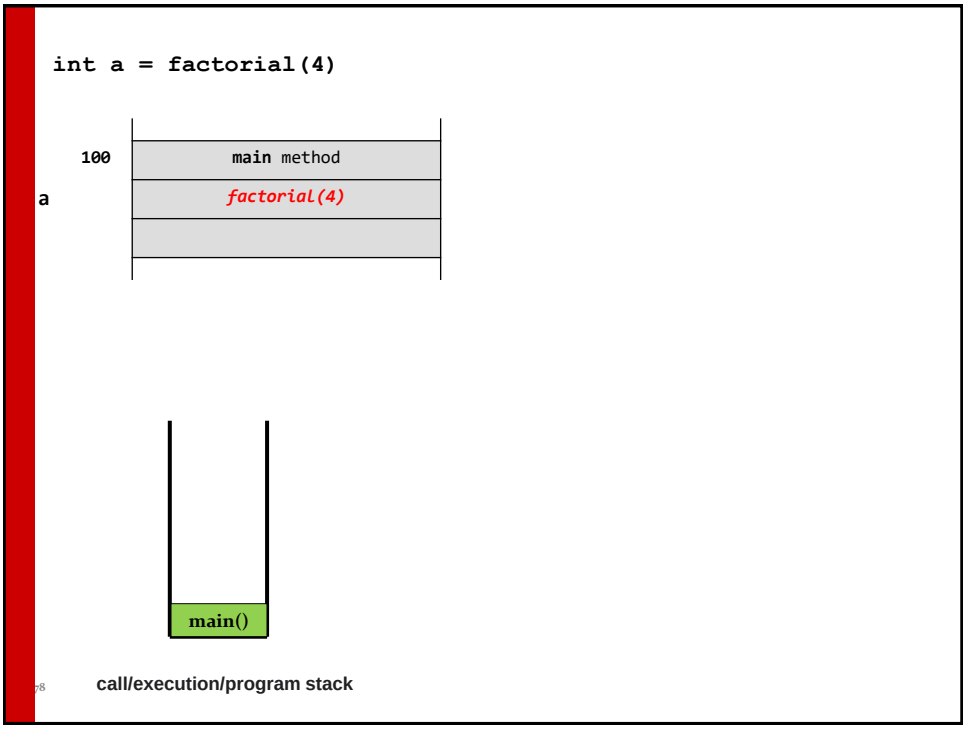
76

```
factorial(5)
--> 5 * factorial(4)
--> 5 * 4 * factorial(3)
--> 5 * 4 * 3 * factorial(2)
--> 5 * 4 * 3 * 2 * factorial(1)
--> 5 * 4 * 3 * 2 * 1 * factorial(0)
--> 5 * 4 * 3 * 2 * 1 * 1
--> 120
```

76

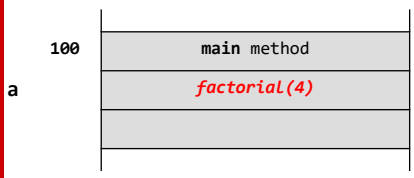

Program trace, more details
static int factorial (4)
{
if(4 == 0) /* base case */
return 1;
return 4 * factorial (4 – 1);
}
static int factorial (3)
{
if(3 == 0) /* base case */
return 1;
return 3 * factorial (3 – 1);
}
static int factorial (2)
{
if(2 == 0) /* base case */
return 1;
return 2 * factorial (2 – 1);
}
static int factorial (1)
{
if(1 == 0) /* base case */
return 1;
return 1 * factorial (1 – 1);
}
static int factorial (0)
{
if(0 == 0) /* base case */
return 1;
return n * factorial (0 – 1);
6
24
2
1
1
int a = factorial(4)

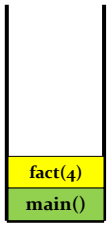
77

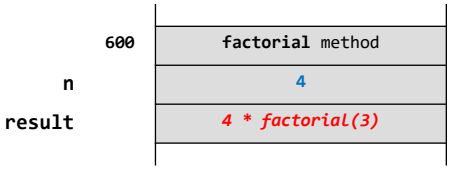

int a = factorial(4)
100
main method
a
factorial(4)
main()
call/execution/program stack

78

```
int a = factorial(4)
```





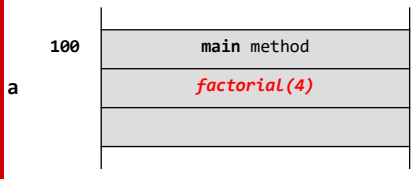


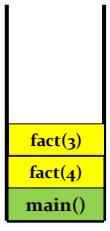
call/execution/program stack

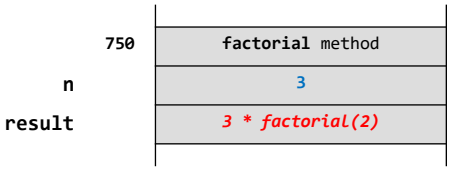
79

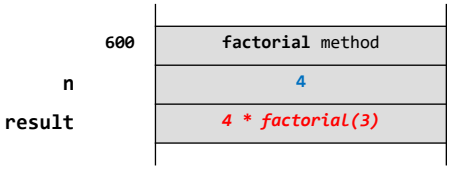
79

```
int a = factorial(4)
```





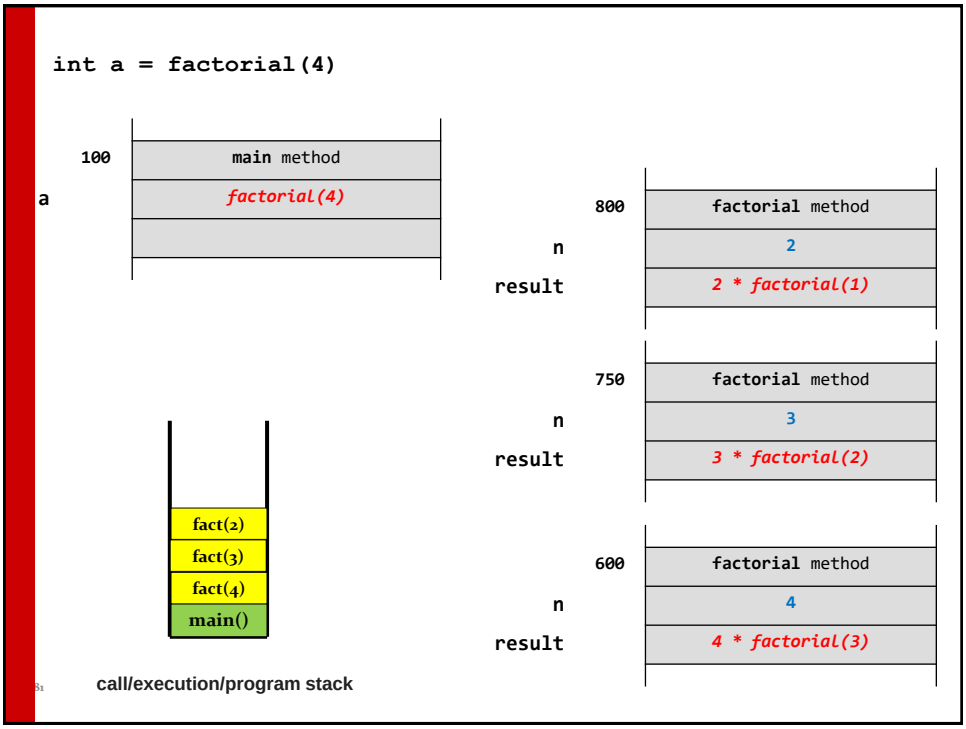




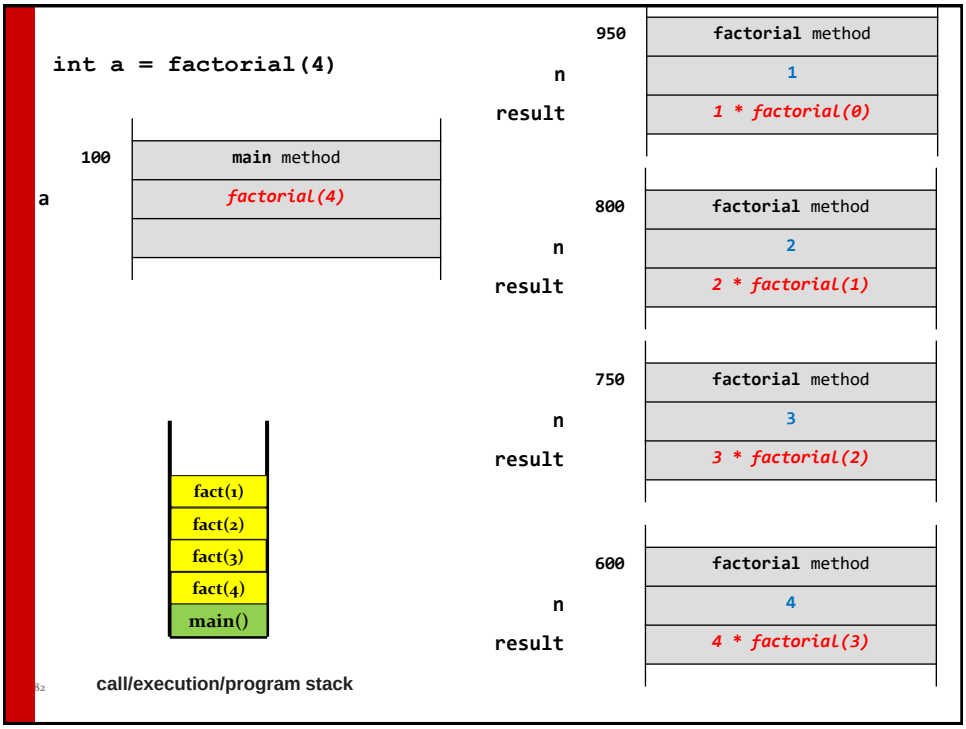
call/execution/program stack

80

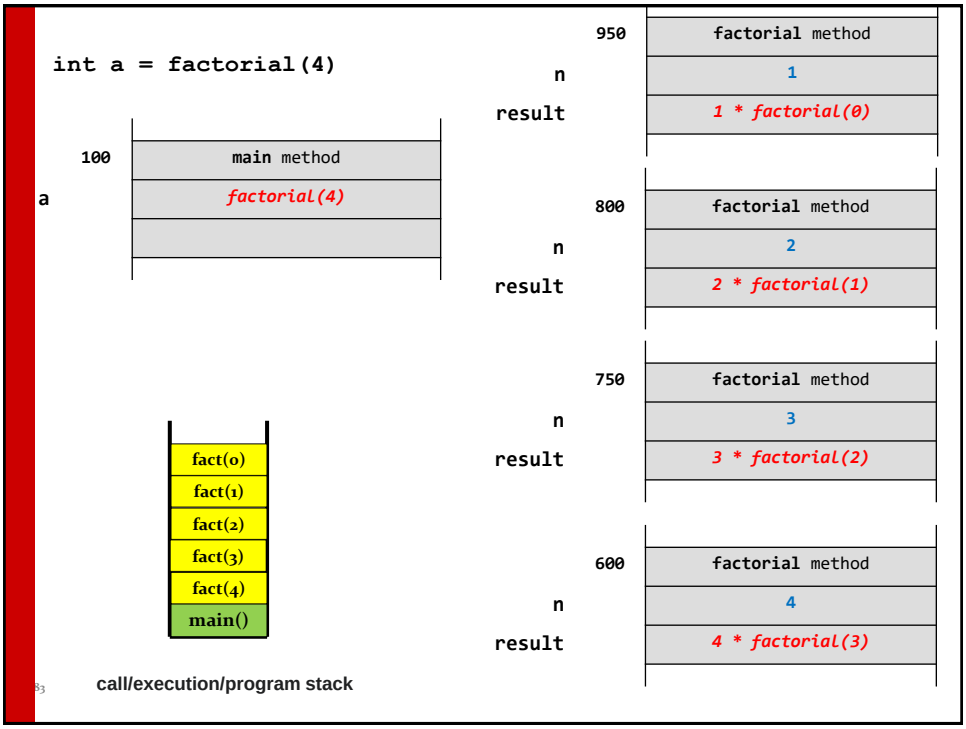
80


int a = factorial(4)
100
main method
a
factorial(4)
800
factorial method
n
2
result
2 * factorial(1)
750
factorial method
n
3
result
3 * factorial(2)
600
factorial method
n
4
result
4 * factorial(3)
fact(2)
fact(3)
fact(4)
main()
call/execution/program stack

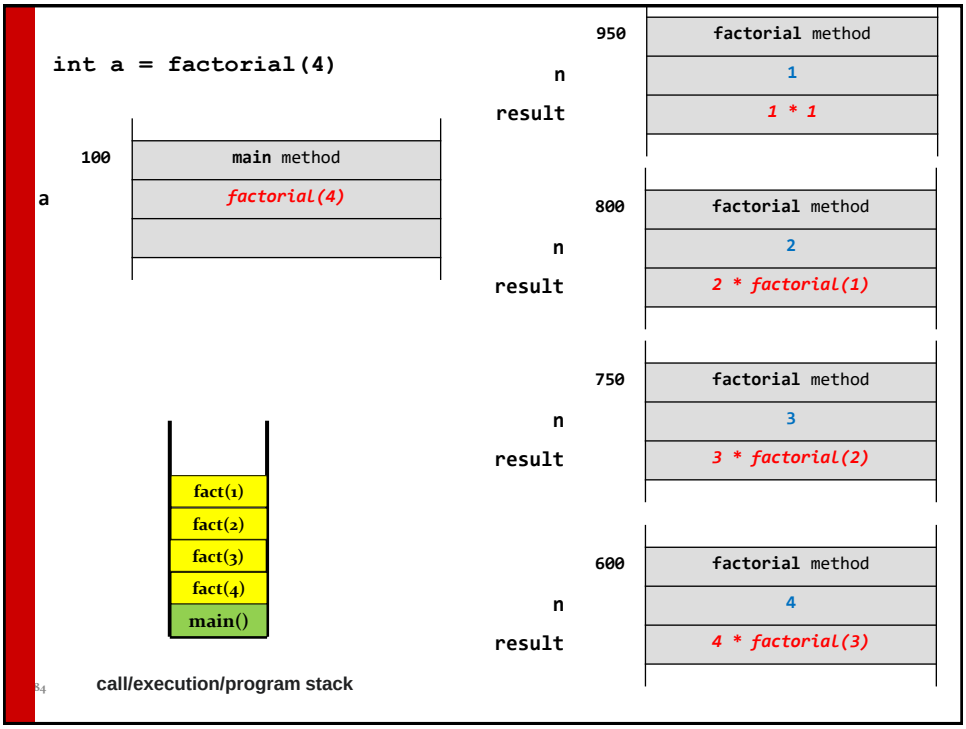
81


int a = factorial(4)
950
factorial method
n
1
result
1 * factorial(0)
100
main method
a
factorial(4)
800
factorial method
n
2
result
2 * factorial(1)
750
factorial method
n
3
result
3 * factorial(2)
600
factorial method
n
4
result
4 * factorial(3)
fact(1)
fact(2)
fact(3)
fact(4)
main()
call/execution/program stack

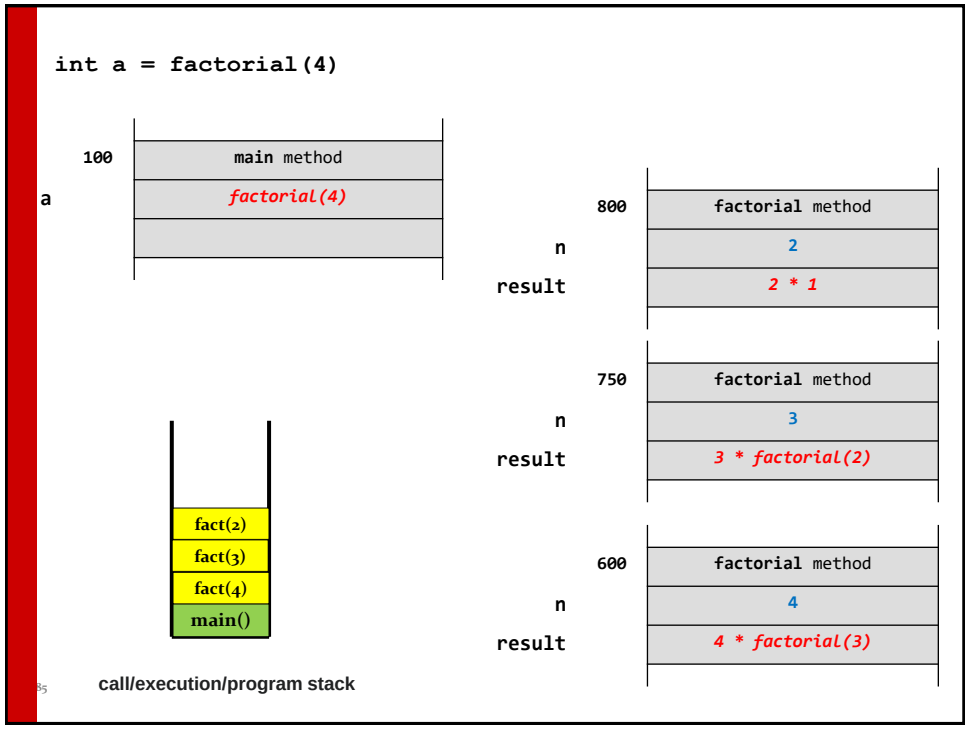
82


int a = factorial(4)
100
main method
a
factorial(4)
fact(0)
fact(1)
fact(2)
fact(3)
fact(4)
main()
call/execution/program stack
950
factorial method
n
1
result
1 * factorial(0)
800
factorial method
n
2
result
2 * factorial(1)
750
factorial method
n
3
result
3 * factorial(2)
600
factorial method
n
4
result
4 * factorial(3)

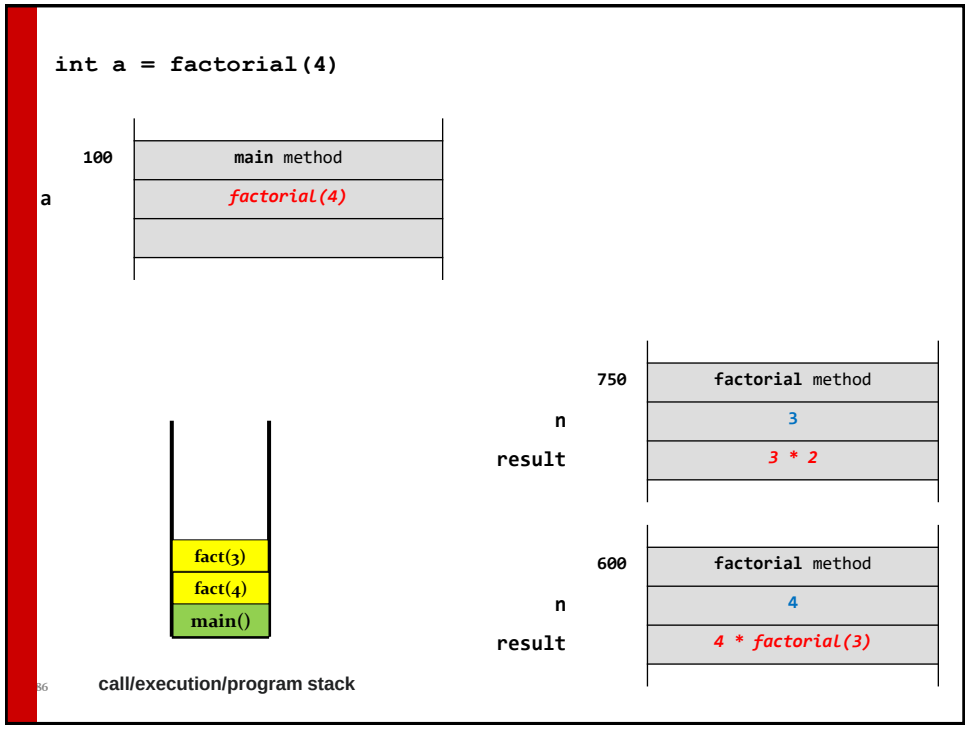
83


int a = factorial(4)
100
main method
a
factorial(4)
fact(1)
fact(2)
fact(3)
fact(4)
main()
call/execution/program stack
950
factorial method
n
1
result
1 * 1
800
factorial method
n
2
result
2 * factorial(1)
750
factorial method
n
3
result
3 * factorial(2)
600
factorial method
n
4
result
4 * factorial(3)

84

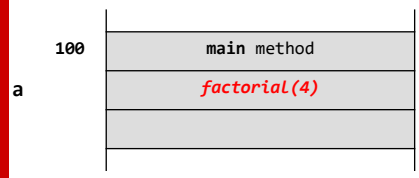

int a = factorial(4)
100
main method
a
factorial(4)
800
factorial method
n
2
result
2 * 1
750
factorial method
n
3
result
3 * factorial(2)
600
factorial method
n
4
result
4 * factorial(3)
fact(2)
fact(3)
fact(4)
main()
85
call/execution/program stack

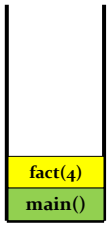
85

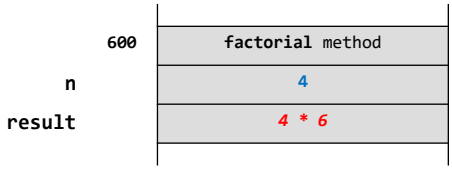

int a = factorial(4)
100
main method
a
factorial(4)
750
factorial method
n
3
result
3 * 2
600
factorial method
n
4
result
4 * factorial(3)
fact(3)
fact(4)
main()
86
call/execution/program stack

86

```
int a = factorial(4)
```





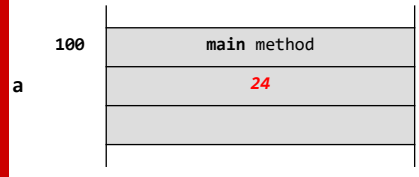


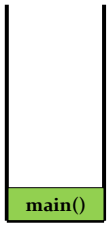
call/execution/program stack

87

87

```
int a = factorial(4)
```





call/execution/program stack

88

88

Recursion

"To get into my house
I must get the key from a smaller house

- C supports recursion
- Think/define recursively

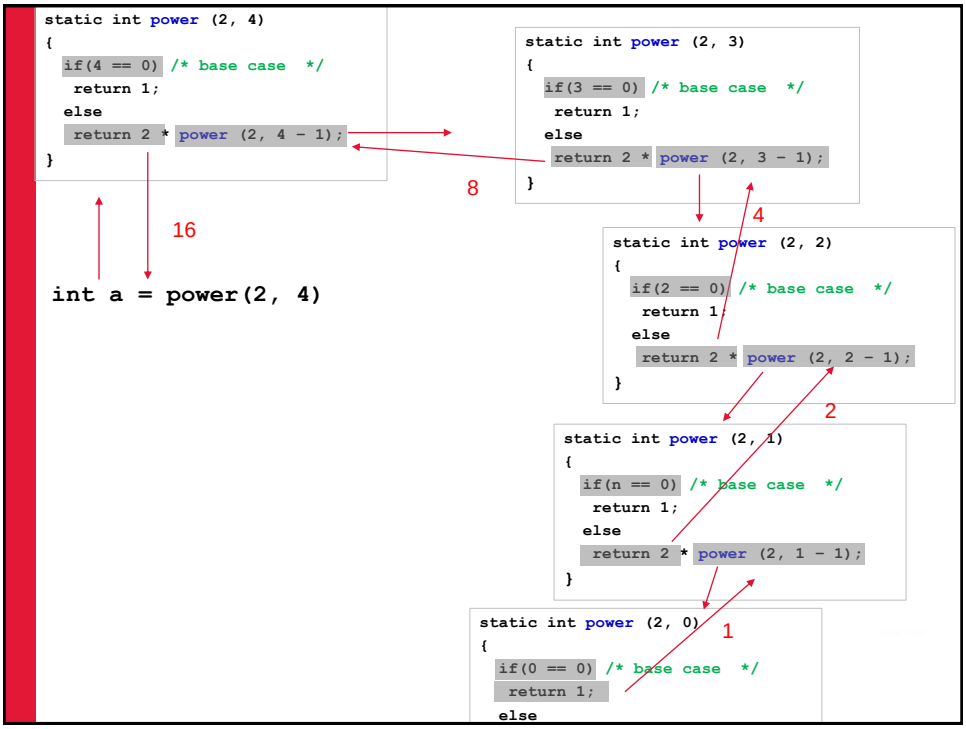
$b^n = b * b^{n-1}$

$$power(base, n) = \begin{cases} 1 & \text{if } n = 0 \\ base \cdot power(base, n-1) & otherwise \end{cases}$$

```
int power (int base, int n)   // assume n >= 0
{
  if(num == 0) /* base case  */
    return 1;
  else
    return base * power (base, n-1);
}
```

89

89



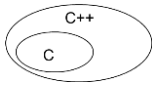
90

- Finished Ch1 – 4

- Other C materials before pointer
  - Common library functions [Appendix of K+R]
  - 2D array, string manipulations

Common library functions [Appendix of K+R]

Included in C++ e.g.,
`<cstring.h>` `<cmath.h>`



<stdio.h>
printf()
scanf()
getchar()
putchar()

sscanf()
sprintf()

gets() puts()
fgets() fputs()

fprintf()
fscanf()

<string.h>
strlen(s)
strcpy(s,s)
strcat(s,s)
strcmp(s,s)
strtok(s,s)

<math.h>
sin() cos()
exp()
log()
pow()
sqrt()
ceil()
floor()

<stdlib.h>
int atoi(s)
double atof(s)
long atol(s)
void rand()
void system()
void exit()
int abs(int)

<assert.h>
assert()

<limit.h>

.....

<ctype.h>
int islower(int)
int isupper(int)
int isdigit(int)
int isxdigit(int)
int isalpha(int)

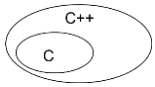
int tolower(int)
int toupper(int)

<signal.h>

<time.h>

Common library functions [Appendix of K+R]

Included in C++ e.g.,
<cstring.h> <cmath.h>



<stdio.h>
printf()
scanf()
getchar()
putchar()

sscanf()
sprintf()

gets() puts()
fgets() fputs()

fprintf()
fscanf()

<string.h>
strlen(s)
strcpy(s,s)
strcat(s,s)
strcmp(s,s)
strtok(s,s)

<math.h>
sin() cos()
exp()
log()
pow()
sqrt()
ceil()
floor()

<stdlib.h>

int atoi(s)
double atof(s)
long atol(s)
void rand()
void system()
void exit()
int abs(int)

<assert.h>
assert()

<limit.h>

<ctype.h>

int islower(int)
int isupper(int)
int isdigit(int)
int isxdigit(int)
int isalpha(int)

int tolower(int)
int toupper(int)

<signal.h>

<time.h>

.....

93

String library functions

- Defined in standard library, prototype in **<string.h>**

- **unsigned int strlen(s)**
  - # of chars before **first** '\0'
  - not counting first '\0'

```
strlen("hello");  // 5
```

H	e	l	\0	o	\0		

```
//strlen?
```

```
int len = strlen(line);
printf("%d\n",strlen(line));
```

```
indigo 324 % gcc strlen.c -Wall
strlen.c: In function 'main':
strlen.c:9:3: warning: format '%d' expects argument of type 'int', but
 argument 2 has type 'size_t' [-Wformat=]
   printf("Hello, world %d\n", strlen(a));     // okay
   ^
```

```
printf("%zu\n",strlen(line));  // not required
```

98

Side notes: these 3 warnings that are ok for this course

```
indigo 316 % gcc strlen0.c -Wall
strlen0.c:6:2: warning: return type defaults to 'int' [-Wreturn-type]
  main()
  ^                                                    // okay
strlen0.c: In function 'main':
strlen0.c:14:1: warning: control reaches end of non-void function [-Wr
eturn-type]
 }                                                     // okay
 ^
```

```
indigo 324 % gcc strlen.c -Wall
strlen.c: In function 'main':
strlen.c:9:3: warning: format '%d' expects argument of type 'int', but
 argument 2 has type 'size_t' [-Wformat=]
   printf("Hello, world %d\n", strlen(a));          // okay
   ^
```

other warnings are not ok, need to fix

```
indigo 319 % gcc strlen0.c
strlen0.c: In function 'main':
strlen0.c:9:3: warning: incompatible implicit declaration of built-in
function 'printf' [enabled by default]
   printf("Hello, world %d\n", strlen(a));          // not okay
   ^
strlen0.c:9:31: warning: incompatible implicit declaration of built-in
 function 'strlen' [enabled by default]
   printf("Hello, world %d\n", strlen(a));          // not okay
                               ^
```

99

String library functions

- Defined in standard library, prototype in **<string.h>**

- **unsigned int strlen(s)**
  - **# of chars before first '\0'**
  - **not counting first '\0'**

```
strlen("hello");  // 5
```

H	e	l	\0	o	\0		

//strlen?

- **strcpy (dest, src)**
  - **strncpy(dest, src, n)**
  - **modify dest**

dest = src ✗

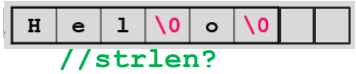
100

String library functions

- Defined in standard library, prototype in `<string.h>`

- `unsigned int strlen(s)`
  - `# of chars before first '\0'`
  - `not counting first '\0'`

`strlen("hello"); // 5`



`//strlen?`

- `strcpy (dest, src)`
  - `strncpy(dest, src, n)`
  - `modify dest`

`dest = src` ✗

- `strcat(s1, s2)` `s1 → s1s2` `s1 + s2` ✗
  - `strncat (s1, s2, n)`
  - `modify s1` `append s2 to the end of s1`

101

String library functions

- Defined in standard library, prototype in `<string.h>`

- `unsigned int strlen(s)`
  - `# of chars before first '\0'`
  - `not counting first '\0'`

`strlen("hello"); // 5`

`//strlen?`

- `strcpy (dest, src)`
  - `strncpy(dest, src, n)`
  - `modify dest`

`dest = src` ✗

- `strcat(s1, s2)` `s1 → s1s2` `s1 + s2` ✗
  - `strncat (s1, s2, n)`
  - `modify s1` `append s2 to the end of s1`

- `int strcmp(s1, s2)` `0 if equal` `<0 if s1<s2, >0 if s1>s2`
  - `strncmp(s1, s2, n)`

lexicographical order

102

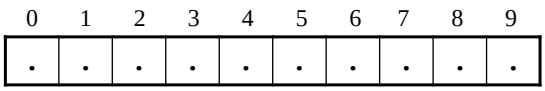
102

```
char str1[] = "NEW";
char str2[] = "YORK";
strcpy(str1, str2);
```

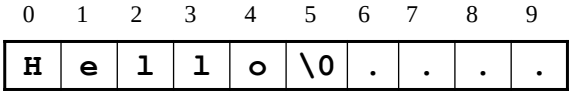
The strcpy() function copies str2 to str1, up to and including str2's null character.

strcpy Compensate for the fact that cannot use = to copy strings

To get from another string (literal) **<string.h>**

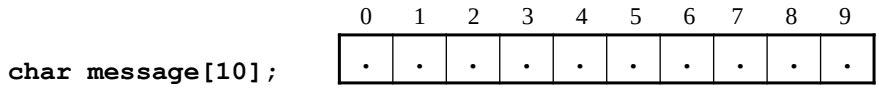



```
char message[10];
strcpy(message, "hello")    // or a variable with "hello"
```

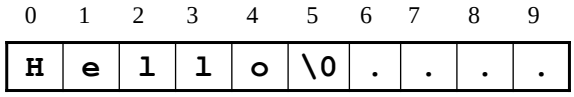



```
strlen(message)? 5  sizeof message? 10  message[4]? 'o'
printf("%s", message)?
strcpy(message , "OK");   ?
```

strcpy Compensate for the fact that cannot use = to copy strings
To get from another string (literal) **<string.h>**

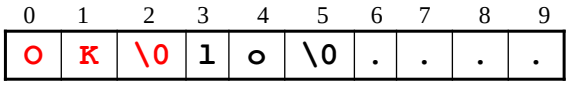



```
char message[10];
strcpy(message, "hello")
```




```
strlen(message)? 5   sizeof message? 10   message[4]? 'o'
printf("%s", message)?
```

```
strcpy(message , "OK");    ?
```

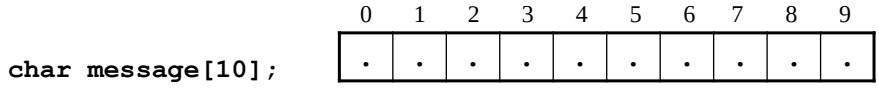



```
strlen(message)?     sizeof message?      message[4]?
printf("%s", message)?
```

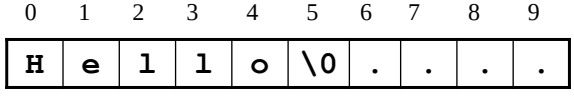
105

105

strncpy Compensate for the fact that cannot use = to copy strings
To get from another string (literal) **<string.h>**




```
char message[10];
strcpy(message, "hello")
```



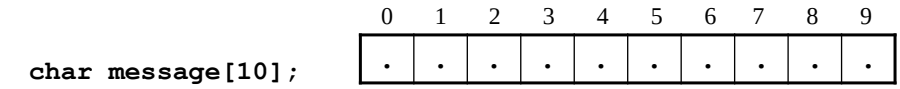

```
strlen(message)? 5   sizeof message? 10   message[4]? 'o'
printf("%s", message)?
```

```
strncpy(message , "OK",3);    ?
```

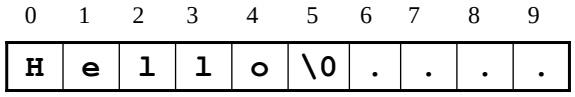
106

strncpy Compensate for the fact that cannot use = to copy strings

To get from another string (literal) **<string.h>**

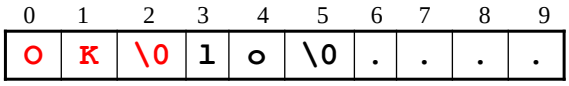



```
char message[10];
strcpy(message, "hello")
```




```
strlen(message)? 5   sizeof message? 10   message[4]? 'o'
printf("%s", message)?
```

```
strncpy(message , "OK",3);    ?
```



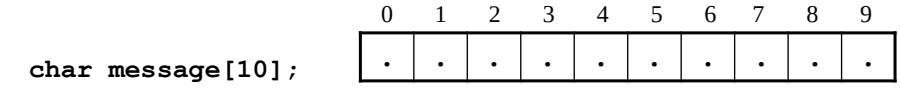

```
strlen(message)?     sizeof message?      message[4]?
printf("%s", message)?
```

107

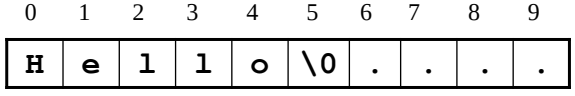
107

strncpy Compensate for the fact that cannot use = to copy strings

To get from another string (literal) **<string.h>**




```
char message[10];
strcpy(message, "hello")
```



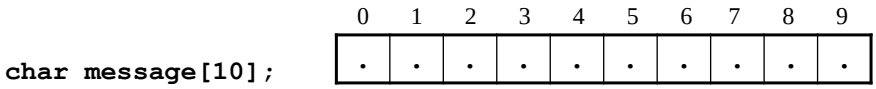

```
strlen(message)? 5   sizeof message? 10   message[4]? 'o'
printf("%s", message)?
```

```
strncpy(message , "OK",2);    ?
```

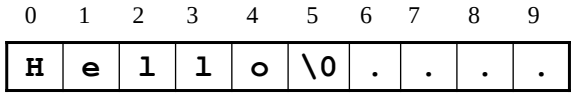
108

strncpy Compensate for the fact that cannot use = to copy strings

To get from another string (literal) **<string.h>**

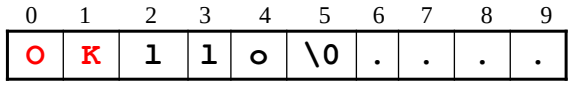



```
char message[10];
strcpy(message, "hello")
```




```
strlen(message)? 5   sizeof message? 10   message[4]? 'o'
printf("%s", message)?
```

```
strncpy(message , "OK",2);    ?
```



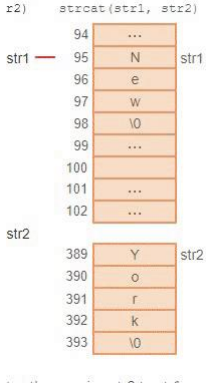
No \0 added

```
strlen(message)?     sizeof message?      message[4]?
printf("%s", message)?
```

109

109

```
char str1[] = "NEW";
char str2[] = "YORK";
strcat(str1, str2);
```

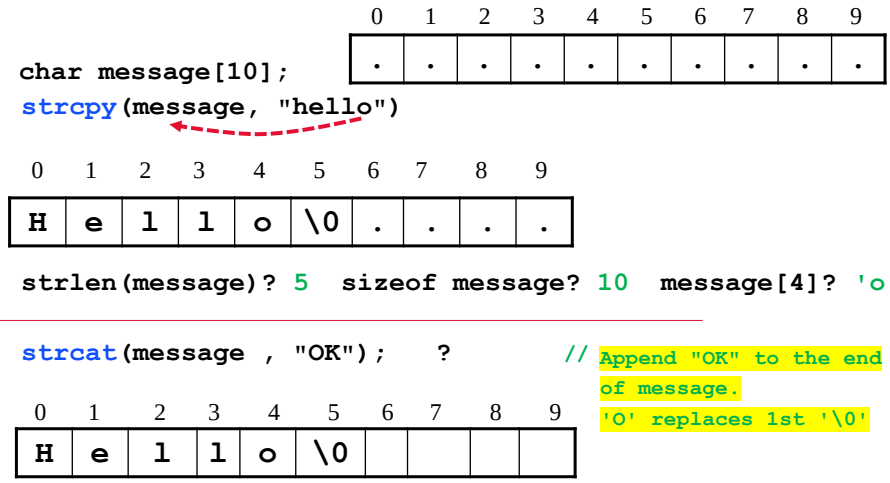


The strcat() function starts at str1's null character, then copies str2 to str1, up to and including str2's null character.

112

112

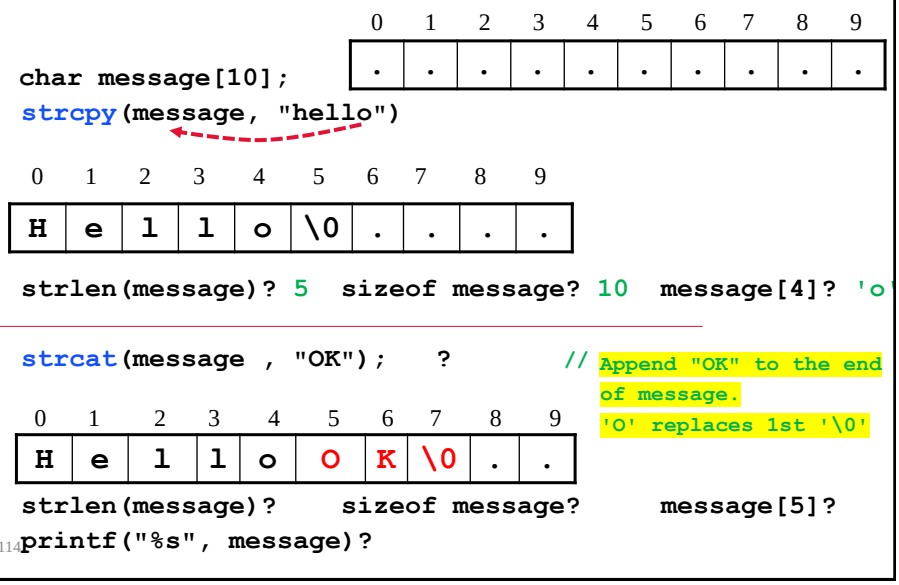
strcat Compensate for fact that can't use + to concatenate strings
To get from another string (literal) **<string.h>**



11

113

strcat Compensate for fact that can't use + to concatenate strings
To get from another string (literal) **<string.h>**

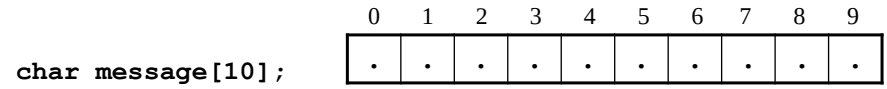


114

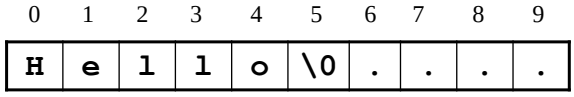
114

strncat Compensate for fact that can't use + to concatenate strings

To get from another string (literal) **<string.h>**

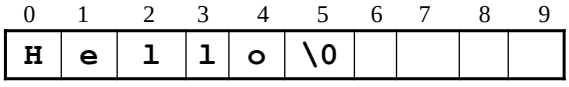



```
char message[10];
strcpy(message, "hello")
```




```
strlen(message)? 5   sizeof message? 10   message[4]? 'o'
```

```
strncat(message , "OK", 1);    ?
```

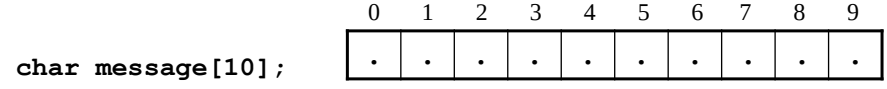


11

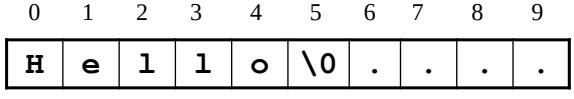
115

strncat Compensate for fact that can't use + to concatenate strings

To get from another string (literal) **<string.h>**

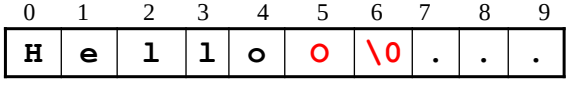



```
char message[10];
strcpy(message, "hello")
```




```
strlen(message)? 5   sizeof message? 10   message[4]? 'o'
```

```
strncat(message , "OK", 1);    ?
```



\0 added

```
strlen(message)?    sizeof message?    message[5]?
printf("%s", message)?
```

116

strcat Compensate for the fact that cannot use + to glue strings

Another example

0	1	2	3	4	5	6	7	8	9
O	K	\0	s		G	O	\0	.	.

```
strcat(message, "Hi")
```

0	1	2	3	4	5	6	7	8	9
O	K	H	i	\0	G	O	\0	.	.

Append "Hi" to the end of message.
'H' replaces 1st '\0'

```
strlen(message)? 4   sizeof message? 10  message[6]? 'o'
printf("%s", message)?  OKHi
```

```
strcat(message , "B");   ?
```

0	1	2	3	4	5	6	7	8	9
O	K	H	i	B	\0	O	\0	.	.

```
strlen(message)? 5   sizeof message? 10   message[6]? 'o'
printf("%s", message)? OKHiB          strncat(mes,"Bye",1)?
```

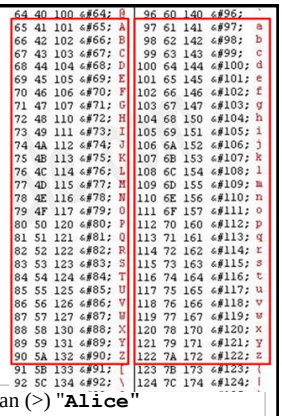


```
#include <stdio.h>
#include <string.h>

int main ()
{
   char str1[15];
   char str2[15];
   int ret;

   strcpy(str1, "abcdef");
   strcpy(str2, "ABCDEF");

   ret = strcmp(str1, str2);

   if(ret < 0)
   {
      printf("str1 is less than str2");
   }
   else if(ret > 0)  // str1 greater than str2
   {
      printf("str2 is less than str1");
   }
   else   // res == 0
   {
      printf("str1 is equal to str2");
   }

   return(0);
}
```

```
int strcmp(s1, s2)
/*  lexicographical order  */
```

```
.... . . . . . .
.... . . . . . .
a b c d e f \0  . . . . .
A B C D E F \0  . . . . .
```

sizeof? 15 strlen? 6



"**abcde**" is greater than (>) "**Alice**"
because 'a' located after 'A' in ASCII table
strcmp("abcded","Alice") returns +

"HelloWorld" < **"Hellothwre"**
because 'W' precedes 't' in ASCII table
strcmp("HelloWorld","Hellothwre") -

"Hello!" equals **"Hello!"** 0
"base" < **"baseball"** -

Same as Java s.compareTo(s2)

```
int strcmp(s1, s2);
```

0 if equal !0 if not equal <0 if s1<s2, >0 if s1>s2

```
int isQuit (char arr[]){
 int i;
 if (arr[0]=='q' && arr[1]=='u' && arr[2]=='i' && arr[3]=='t' &&
    return 1;                                                  )
  else return 0; }
```

```
isQuit(char arr[]){
  if ( strcmp(arr, "quit") == 0 )
    return 1;            // equal
  else return 0
}
```

```
while ( strcmp (arr, "quit") !=0 )
  …..                    // not equal
```

or

```
while ( strcmp (arr, "quit") )
  …..                    // not equal
```

```
while (1) {
  if ( strcmp (arr, "quit")==0)
    break;               // equal
```

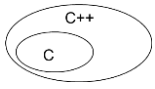
```
while (1) {
  if ( ! strcmp (arr, "quit") )
    break;               // equal
```

119

119

Common library functions [Appendix of K+R]

Included in C++ e.g.,
<cstring.h> <cmath.h>



<stdio.h>
printf()
scanf()
getchar()
putchar()

sscanf()
sprintf()

gets() puts()
fgets() fputs()

fprintf()
fscanf()

<string.h>
strlen(s)
strcpy(s,s)
strcat(s,s)
strcmp(s,s)
strtok(s,s)

<math.h>
sin() cos()
exp()
log()
pow()
sqrt()
ceil()
floor()

<stdlib.h>

int atoi(s)
double atof(s)
long atol(s)
void rand()
void system()
void exit()
int abs(int)

<assert.h>
assert()

<limit.h>

.....

<ctype.h>

int islower(int)
int isupper(int)
int isdigit(int)
int isxdigit(int)
int isalpha(int)

int tolower(int)
int toupper(int)

<signal.h>

<time.h>

122

character library functions

- Defined in standard library, prototype in **<ctype.h>**

- **int islower(int ch)** `ch >='a' && ch <='z'`
- **int isupper(int ch)** `ch >='A' && ch <='Z'`
- **int isalpha(int ch)** `islower(ch) || isupper(ch)`
- **int isdigit(int ch)** `ch >='0' && ch <='9'`
- **int isalnum(int ch)** `isalpha(ch) or isdigit(ch)`
- **int isxdigit(int ch)** `'0'-'9', 'a'-'f','A'-'F',`

- **int tolower(int ch)**
- **int toupper(int ch)**

```
if (isupper(ch))
   return ch + ('a' - 'A');
else return ch;
```

ch not changed

123

YORK UNIVERSITÉ UNIVERSITY

123

Example

```
#include<stdio.h>

/* copying input to output with
upper-case converted to lower-case letters */
main(){
    int c;
    c= getchar();
    while (c != EOF)
    {
        if (c >= 'A' && c <= 'Z')
            c += 'a'- 'A';

        putchar(c);

        c = getchar();
    }
    return 0;
}
```

64	40	100	@	@	96	60	140	`	`
65	41	101	A	A	97	61	141	a	a
66	42	102	B	B	98	62	142	b	b
67	43	103	C	C	99	63	143	c	c
68	44	104	D	D	100	64	144	d	d
69	45	105	E	E	101	65	145	e	e
70	46	106	F	F	102	66	146	f	f
71	47	107	G	G	103	67	147	g	g
72	48	110	H	H	104	68	150	h	h
73	49	111	I	I	105	69	151	i	i
74	4A	112	J	J	106	6A	152	j	j
75	4B	113	K	K	107	6B	153	k	k
76	4C	114	L	L	108	6C	154	l	l
77	4D	115	M	M	109	6D	155	m	m
78	4E	116	N	N	110	6E	156	n	n
79	4F	117	O	O	111	6F	157	o	o
80	50	120	P	P	112	70	160	p	p
81	51	121	Q	Q	113	71	161	q	q
82	52	122	R	R	114	72	162	r	r
83	53	123	S	S	115	73	163	s	s
84	54	124	T	T	116	74	164	t	t
85	55	125	U	U	117	75	165	u	u
86	56	126	V	V	118	76	166	v	v
87	57	127	W	W	119	77	167	w	w

```
    c= getchar();
    while (c != EOF)
    {
        if (isupper(c))
            tolower(c);

        putchar(c);

        c = getchar();
    }
    return 0;
}
```

java.lang.Character

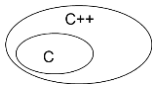
```
public static boolean isUpperCase(char ch)
public static boolean isLowerCase(char ch)
public static char toUpperCase(char ch)
public static char toLowerCase(char ch)
```

124

124

Common library functions [Appendix of K+R]

Included in C++ e.g.,
<cstring.h> <cmath.h>



<stdio.h>
printf()
scanf()
getchar()
putchar()

sscanf()
sprintf()

gets() puts()
fgets() fputs()

fprintf()
fscanf()

<string.h>
strlen(s)
strcpy(s,s)
strcat(s,s)
strcmp(s,s)
strtok(s,s)

<math.h>
sin() cos()
exp()
log()
pow()
sqrt()
ceil()
floor()

<stdlib.h>

int atoi(s)
double atof(s)
long atol(s)
void rand()
void system()
void exit()
int abs(int)

<assert.h>
assert()

<limit.h>

.....

<ctype.h>

int islower(int)
int isupper(int)
int isdigit(int)
int isxdigit(int)
int isalpha(int)

int tolower(int)
int toupper(int)

<signal.h>

<time.h>

125

Utility library functions: number conversion …

- Defined in standard library, prototype in **<stdlib.h>**

- `int atoi("string" s)` "6"
- `double atof("string" s)` "3.24"
- `long atol ("string" s)`
- `int rand(void) void srand(unsigned seed)`
- `void abort(void)`
- `void exit(int)` EXIT_SUCESS, EXIT_FAILURE
- `int system(commandString)` ⇨
- `int abs(int) long labs(long)`
- `void qsort(……) // quick sort`

- **malloc, calloc, free**

126

YORK UNIVERSITÉ UNIVERSITY

126

C and Unix are closely related

```
#include<stdio.h>

int main()
{
   system("ls -l");   // execute unix command line ls -l

   system("mkdir xxx"); // execute unix command line mkdir xxx


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

   system("ls -l");
}
```

```
total 0
drwx------ 2 huiwang faculty 6 Jan 28 23:11 dir1
===
total 0
drwx------ 2 huiwang faculty 6 Jan 28 23:11 dir1
drwx------ 2 huiwang faculty 6 Jan 28 23:11 xxx
```

127

127

For your information

```
void initializeHardware(void)
{  /* initialize hardware */
   if(connect_robot("/dev/ttyUSB0", 38400) == FALSE){
      fprintf(stderr,"unable to connect to robot\n");

      sprintf(buf, "aplay ./sounds/%s", sth_wrong.wav);
      system(buf); //system("aplay ./sounds/sth_wrong.wav")
                   // execute unix command aplay ./…wav
      exit(EXIT_FAILURE);
   }
   // else connected
   enableSonars();
   system("aplay ./sounds/wakingUp.wav");
}
```

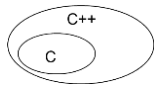
aplay - command-line sound recorder and player for ALSA soundcard driver

128

128

Common library functions [Appendix of K+R]

Included in C++ e.g.,
<cstring.h> <cmath.h>



<stdio.h>
printf()
scanf()
getchar()
putchar()

sscanf()
sprintf()

gets() puts()
fgets() fputs()

fprintf()
fscanf()

<string.h>
strlen(s)
strcpy(s,s)
strcat(s,s)
strcmp(s,s)
strtok(s,s)

<math.h>
sin() cos()
exp()
log()
pow()
sqrt()
ceil()
floor()

<stdlib.h>

int atoi(s)
double atof(s)
long atol(s)
void rand()
void system()
void exit()
int abs(int)

<assert.h>
assert()

<limit.h>

.....

<ctype.h>

int islower(int)
int isupper(int)
int isdigit(int)
int isxdigit(int)
int isalpha(int)

int tolower(int)
int toupper(int)

<signal.h>

<time.h>

Diagnostics library functions

- Defined in standard library, prototype in **<assert.h>**

- **void assert(int *expression*)**

```
int x = -1;
assert(x > 0)
```

print **Assertion failed: *expression*, file *file*, line *lnum***
Then **abort()**

For your information

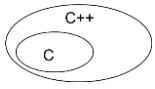
Using the assert() macro.

```
1: /* The assert() macro. */
2:
3: #include <stdio.h>
4: #include <assert.h>
5:
6: main()
7: {
8:     int x;
9:
10:     printf("\nEnter an integer value: ");
11:     scanf("%d", &x);
12:
13:     assert(x >= 0);
14:
15:     printf("You entered %d.\n", x);
16:     return(0);
17: }
Enter an integer value: 10
You entered 10.
Enter an integer value: -1
Assertion failed: x, file list19_3.c, line 13
Abnormal program termination
```

For your information

Common library functions [Appendix of K+R]

Included in C++ e.g.,
<cstring.h> <cmath.h>



<stdio.h>
printf()
scanf()
getchar()
putchar()

sscanf()
sprintf()

gets() puts()
fgets() fputs()

fprintf()
fscanf()

<string.h>
strlen(s)
strcpy(s,s)
strcat(s,s)
strcmp(s,s)
strtok(s,s)

<math.h>
sin() cos()
exp()
log()
pow()
sqrt()
ceil()
floor()

<stdlib.h>

int atoi(s)
double atof(s)
long atol(s)
void rand()
void system()
void exit()
int abs(int)

<assert.h>
assert()

<limit.h>

.....

<ctype.h>

int islower(int)
int isupper(int)
int isdigit(int)
int isxdigit(int)
int isalpha(int)

int tolower(int)
int toupper(int)

<signal.h>

<time.h>

math library functions

- Defined in standard library, prototype in **<math.h>**
- Need to link by **-lm**

- **double sin(double x), cos(x), tan(x)**
- **double asin(x) acos(x) atan(x)** …
- **double exp(x)** e^x
- **double log(x) -- ln(x)**
- **double log10(x)**
- **double pow(x,y)** x^y
- **double sqrt(x)** $\sqrt{x}$
- **double ceil(x)** smallest int not less than x, as a double!
- **double floor(x)** largest int not greater than x, as a double!

x, y are of type double
return double

133 sqrt(9.0) evaluates to 3.0 pow(6.0, 2.0) evaluates to 36.0

YORK UNIVERSITÉ UNIVERSITY

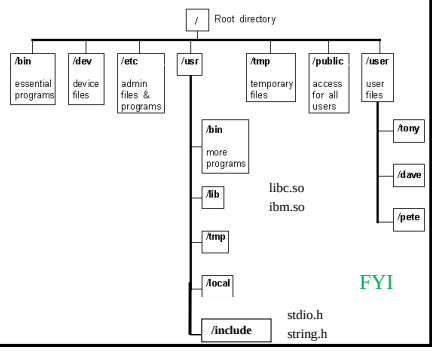
How C Programs are Compiled

- C programs go through three stages to be compiled:
  - Preprocessor - handles #include and #define etc
  - Compiler - converts C code into binary processor instructions ("object code")
  - Linker - puts multiple files together, load library function (e.g. printf, strlen) and creates an executable program

134

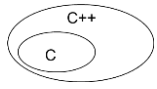
Some complier options

- Option: -l*library*
  - Link with object library **`gcc -lm main.c`**
    - Links math object library (if use **`pow() ceil() sin()`** etc)
    - Needed in the lab
    - Don't forget to use **`#include <math.h>`** at the beginning
  - **`gcc main.c -lp`**
    - Links pthread …..



Common library functions [Appendix of K+R]

Included in C++ e.g.,
`<cstring.h>` `<cmath.h>`



`<stdio.h>`
`printf()`
`scanf()`
`getchar()`
`putchar()`

`sscanf()`
`sprintf()`

`gets()` `puts()`
`fgets()` `fputs()`

`fprintf()`
`fscanf()`

`<string.h>`
`strlen(s)`
`strcpy(s,s)`
`strcat(s,s)`
`strcmp(s,s)`
`strtok(s,s)`

`<math.h>`
`sin()` `cos()`
`exp()`
`log()`
`pow()`
`sqrt()`
`ceil()`
`floor()`

`<stdlib.h>`

```
int    atoi(s)
double atof(s)
long   atol(s)
void   rand()
void   system()
void   exit()
int    abs(int)
```

`<assert.h>`
`assert()`

`<limit.h>`

.....

`<ctype.h>`

```
int islower(int)
int isupper(int)
int isdigit(int)
int isxdigit(int)
int isalpha(int)

int tolower(int)
int toupper(int)
```

`<signal.h>`

`<time.h>`

stdio library functions

- Defined in standard library, prototype in **<stdio.h>**

- **getchar, putchar**
- **scanf, printf**

- **gets, fgets, puts, fputs** **/*read write line */**

/* read from, print to a string */

- **sscanf, sprintf**

- **fscanf, fprintf**

- .....

139

Basic I/O functions **<stdio.h>**

- **int printf (char *format, arg1, ….);**
  - Formats and prints arguments on standard output (screen or > outputFile)
  - **printf("This is a test %d \n", x)**

- **int scanf (char *format, arg1, ….);**
  - Formatted input from standard input (keyboard or < inputFile)
  - **scanf("%d %c", &x, &y)**

- **int sprintf (char * str, char *format, arg1,…..);**
  - Formats and prints arguments to char array (string) str
  - **sprintf(str, "This is a test %d \n", x)** // nothing print on stdout!

- **int sscanf (char * str, char *format, arg1, ….);**
  - Formatted input from char array (string) str
  - **sscanf(str, "%d %c", &x, &y)** // tokenize string str

strings: set /get in general

```
char message[20];
```

- To get from another string (literal) – `strcpy` prototype `<string.h>`
  - `strcpy(message, "hello")`
  - `str[] = "Hi";  strcpy(message , str);` `Hi\0`

`"john 12 2.3"`

- `sprintf` -- Defined in standard library, prototype in `<stdio.h>`
  - `sprintf(message, "%s %d %f", "john",12, 2.3);`
    `"john 12 2.300000"`
    format and then write to message `\0` added
  - `sprintf(message, "%s %d-%.3f", "john",12, 2.3);`
    `"john 12-2.300"`

- `sscanf(message, "%s %d-%f", name, &age, &rate );`
  tokenizing string message

141 Way of generating/tokenizing a string

141

```
#include <stdio.h>

main(){
 char message [30];
 int age =20;   char name[]="john";  double rate = 4.3456;
 printf("%s %d-%f\n", name, age, rate); // john 20-4.345600
 printf("%s %d-%.3f\n", name, age, rate); // john 20-4.346

 // format and write to message
 sprintf(message, "%s %d-%.3f", name, age, rate); // no screen
 printf("%s\n", message); // john 20-4.346

 int age2; float rate2;  char name2[20];

 // tokenize message
 sscanf(message, "%s %d-%f", name2, &age2, &rate2);

 printf("%s\n", name2); // john
 printf("%d\n", age2);  // 20
 printf("%f\n",rate2);  // 4.346000
 printf("%.3f\n",rate2);  // 4.346
```

142

142

- No live lab session today and tomorrow
- Lab4 first part posted today
- SMQ1 this Friday 7pm~3am.
- Assignment1 soon

YORK UNIVERSITÉ UNIVERSITY