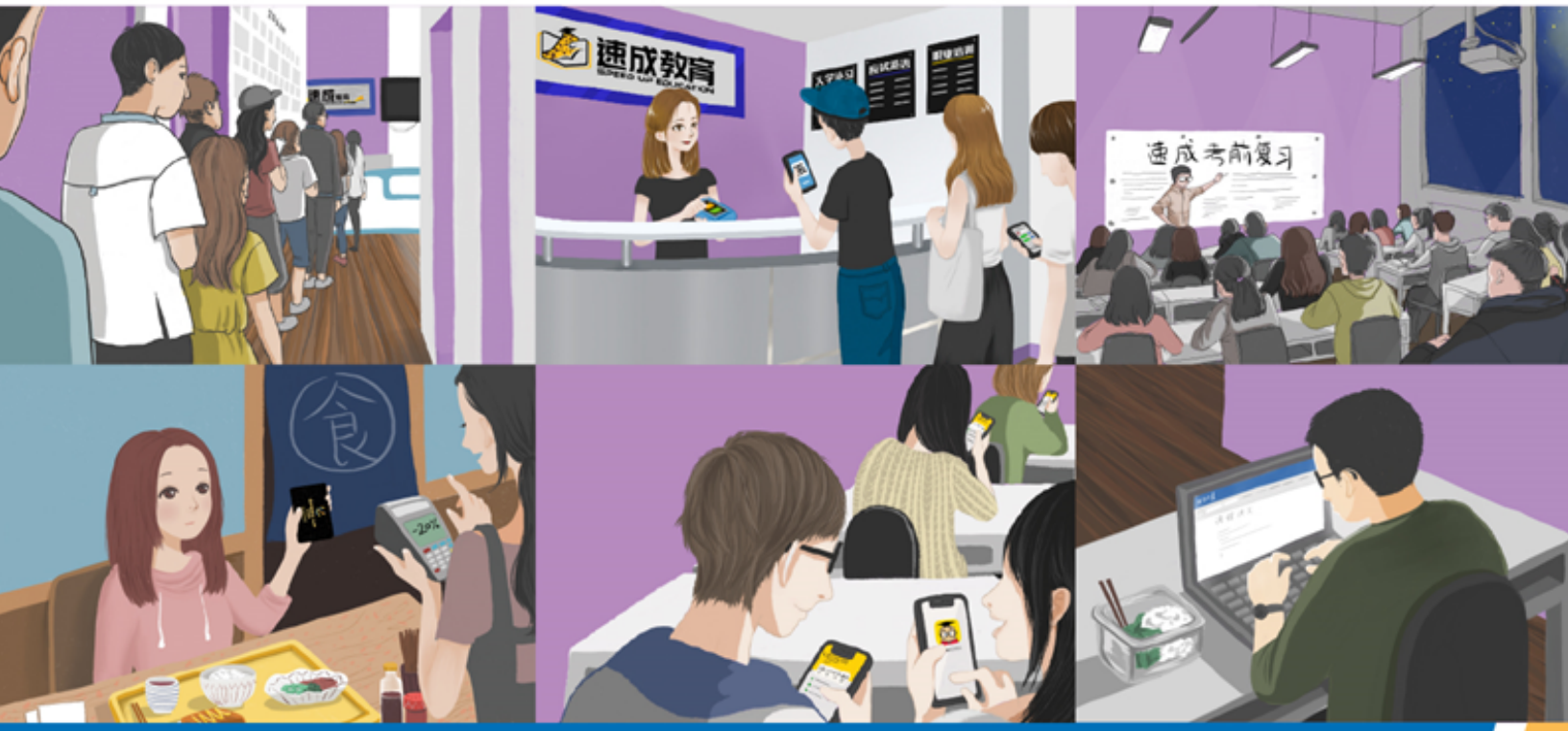
速成教育
SPEED UP EDUCATION




# CSCA48

## Introduction to Computer Science II

导师： **VC**

**UTSC** Week 01 & 02 Class | 2025/1/11

# Steps to Create, Compile, Run a C file

1. Open any text editor (notepad++, sublime test, **VSCode**, .etc)
2. Create a file and save it with the extension .c (ex: week01.c)
3. Once you're done editing your code, **Save your changes**
   **Windows:** press `Ctrl + S`
   **MacOS:** press `Command + S`
4. Open terminal
   **Windows:** press `Win + R`, type `cmd`, press enter
   **MacOS:** press `Command + Space`, type `terminal`, press enter
5. "cd" into the same directory (folder) as your .c file
   **Windows:**
   `dir` – List directory contents
   `cd` – Change working directory
   **MacOS:**
   `ls` – List directory contents
   `cd` – Change working directory
6. type `gcc xxx.c -o yyy` (this compile `xxx.c` and create a executable file called `yyy`)
7. run the executable file yyy
   **Windows:**
   cmd: type `yyy` and press enter
   power shell: type `./yyy` and press enter
   **MacOS:**
   terminal: type `./yyy` and press enter
8. If you Make a change, **re-compile** it (repeat step 3 and step 6)
9. Run it again (repeat step 7)

```c
#include <stdio.h>
int main() {
  printf("Hello World!!");
  return 0;
}
```

week01.c

```
Vincents-MacBook-Pro:~ vincenttse$ ls
SpeedUp          TheClass         Documents        Desktop          Downloads        other..
Vincents-MacBook-Pro:~ vincenttse$ cd SpeedUp/CSCA48/2021W/week01
Vincents-MacBook-Pro:week01 vincenttse$ ls
week01.c
Vincents-MacBook-Pro:week01 vincenttse$ gcc week01.c -o week01
Vincents-MacBook-Pro:week01 vincenttse$
Vincents-MacBook-Pro:week01 vincenttse$ ls
week01.c         week01
Vincents-MacBook-Pro:week01 vincenttse$ ./week01
Hello world!!
Vincents-MacBook-Pro:week01 vincenttse$
```

# Some Basic of C

1. You must include a main function in the .c file you wish to run.
2. The main function must return 0 to tell the computer it has successfully terminated (exit code 0).
3. You must indicate the return type of each function in C. (ex: `int main() { … }` )
4. You must include the library (header) stdio.h in order to use some function like printf
5. When printing, you can use `%_` as placeholders and assign the value.

```
%s

%c

%d

%ld

%f
```

```
printf("my name is %s, and my gpa is %.2f", "Vincent", 4.0)
```

Will give you the ouput: `my name is Vincent, and gpa is 4.00`

# Basic Sample Code in C

```c
#include <stdio.h>

int main() {

    int a = 5;
    long l = 9999999;
    char c = 'A';
    float f = 1.5;
    double d = 3.14159;

    printf("%d\n", a);
    printf("%ld\n", l);
    printf("%c %d\n", c, c);
    printf("%.2f\n", f);
    printf("%.4f\n", d);


    for(int i=0; i<10; i=i+1) {
        printf("%d\n", i);
    }

    for(float j=10000; j>=.00001; j=j/10.0) {
        printf("%f\n", j);
    }

    for(char d='A'; d<'F'; d=d+1) {
        printf("%c , %d\n", d, d);
    }

    // int a=4;
    int x = 4;
    int y = 1;
    if ((x==4) && (y==2)){
        printf("hello\n");
    } else if (1 || (x==3)) {
        printf("hey\n");
    } else {
        printf("bye\n");
    }
    return 0;
}
```

**OutPut:**

```
5
9999999
A 65
1.50
3.1416
0
1
2
3
4
5
6
7
8
9
10000.000000
1000.000000
100.000000
10.000000
1.000000
0.100000
0.010000
0.001000
0.000100
A , 65
B , 66
C , 67
D , 68
E , 69
hey
```

# Variables & Memory

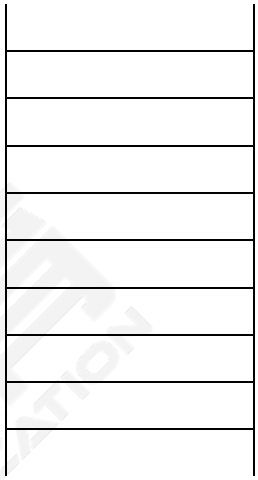
## Python

```
a = 5
c = '1'
```

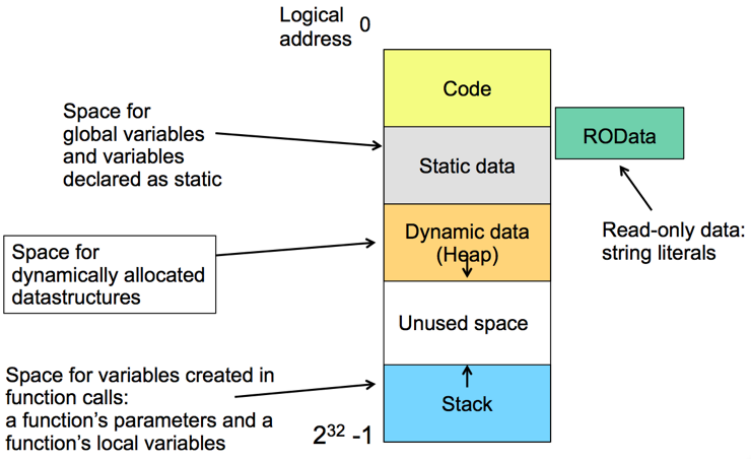
| Variable | Memory |
|---|---|
| | |

## C

```
int main() {
   int a;
   char c;
   a = 5;
   c = '1';
   a ++;
}
```



In CSCA48:

# Pointers

## What is Pointer?

Variables that store address of another variable

Consider the following code:

```
int main() {
    int a;
    int *p;
    p = &a;
    a = 5;
}
```

What happens in the memory when we compile and run the program above?


What will be the output of the following?

```
printf("%p\n", p);
printf("%p\n", &a);
printf("%p\n", &p);
printf("%d\n", *p);
```

Now we add a line of code at the end:

```
*p = 8;
```

What will be the output of the following?

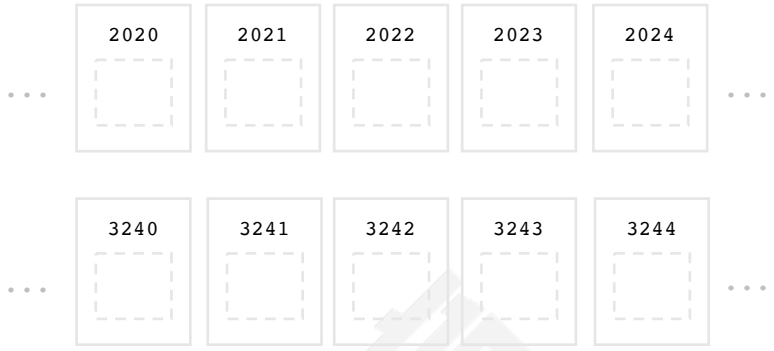
```
printf("%d\n", a);
```

**知识点**

`&a` 得到 a 的地址
`*p` 得到 p 指向的那個地址裡的 value (de-reference)

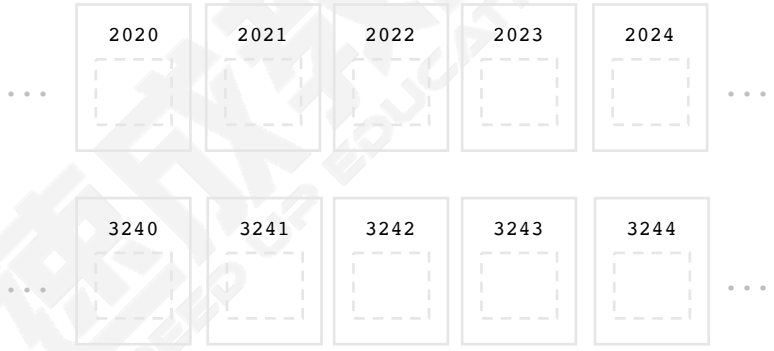
## Why do we need pointers?

```
int increment(int a) {
    int res = a + 1;
    return res;
}

int main () {
    int a = 10;
    a = increment(a);
    printf("%d\n", a);
}
```

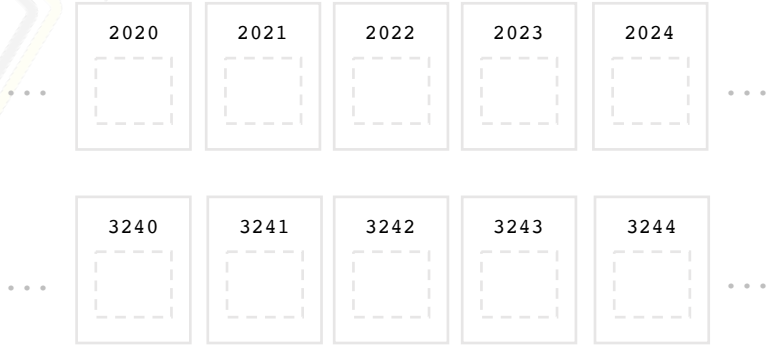



现在把 increment 改成 void, 想让它直接改 main 里的 variable

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

int main () {
    int a = 10;
    increment(a);
    printf("%d\n", a);
}
```

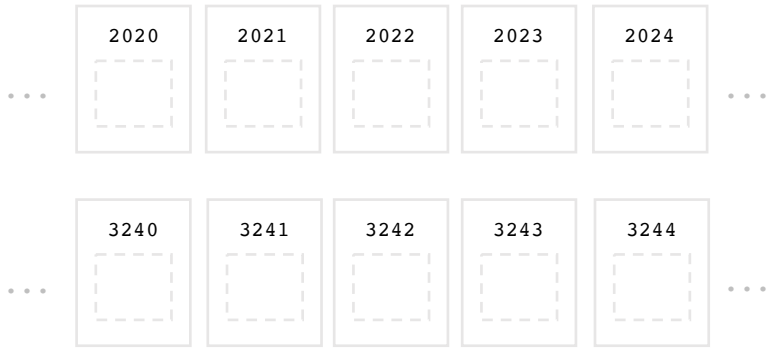



```
void increment(int *p) {
    *p = (*p) + 1;
}

int main () {
    int a = 10;
    increment(&a);
    printf("%d\n", a);
}
```




## 常见的错误

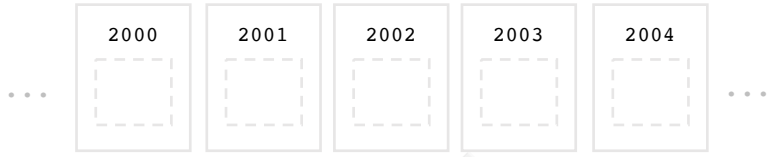
```
int *increment(int a) {
    int res = a + 1;
    return &res;
}

int main () {
    int a = 10;
    int *res = increment(a);
    printf("%d\n", *res);
}
```



## Pointer Arithmetic

What will be the output of the following program?

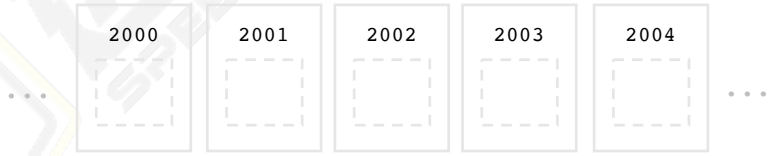
```
#include<stdio.h>
int main() {
  int a = 10;
  int *p;
  p = &a;
  // Pointer arithmetic
  printf("%p\n", p);
  printf("%p\n", p+1);
  printf("%p\n", *(p+1));
  printf("%d\n", *(p+5));
  return 0;
}
```




## Pointer to Pointer

What will be the output of the following program?

```
#include<stdio.h>
int main() {
  int a = 10;
  int *p = &a;
  *p = 6;
  int **q = &p;
  int*** r = &q;
  // int*** r = &p;
  printf("%d\n", *p);
  printf("%d\n", *q);
  printf("%d\n", *(*q));
  printf("%d\n", *(*r));
  printf("%d\n", *(*(*r)));
  ***r = 10;
  printf("a = %d\n", a);
  **q = *p + 2;
  printf("a = %d\n", a);


  return 0;
}
```



# Arrays

```
int main() {
    int my_array[10];  // This create an array that can store 10 integer values

    my_array[0] = 10;  // This is the first entry in the array
    my_array[9] = 5;   // This is the last entry in the array
}
```

What happens in the memory when we compile and run the program above

10 consecutive boxes suitable for holding integer value are reserved and the size of array is FIXED.
The reserved entries won't be assigned to any other variables, parameters, or return values.
**IMPORTANT**, the reserved entries contain junk until you have initialized them to some value.

What will be the output of: `printf("%d\n", my_array[3])`

**IMPORTANT**, C will not warn you or protect you from accessing array entries outside valid index.

What will happen if we do: `my_array[10] = 5;`

**If your program is behaving in unexpected ways, check your array indexing and make sure you don't have indexes out of bounds at any point!**

## 練習：Array & Pointer

What should be printed ?

```
int A[5] = {2, 4, 5, 8, 1};
int *p = A;

printf("%p", &A);
printf("%p", A);

printf("%d", A[0]);
printf("%d", *(A+0));
printf("%d", *A);

printf("%d", A[2]);
printf("%d", *(A+2));

printf("%p", &p);
printf("%p", p);

printf("%d", *p);
printf("%d", p[0]);

printf("%d", *(p+3));
printf("%d", p[3]);

p = p+2;
printf("%d", p[1]);

A = A+1;
```

## Remainder:

Address: `&A[i] or (A+i)`

Value: `A[i] or *(A+i)`

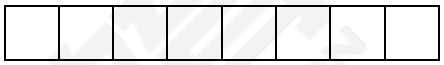
# Character Arrays (String)

## Strings – group of characters

- Size of array >= number of characters in string + 1
- John"  size >= 5
- ~~Example 1:~~

```
char name[4];
name[0] = 'J';  name[1] = 'o';  name[2] = 'h';  name[3] = 'n';
```

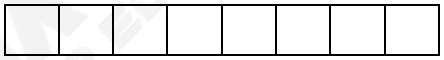
- ~~Example 2:~~



```
char name[8];
name[0] = 'J';  name[1] = 'o';  name[2] = 'h';  name[3] = 'n';
printf("%s\n", name)  <- How would the it knows the string is only up till index 3?
```

Solution: We put a null character at the end of the string '\0'

- Example 3:



```
char name[8];
name[0] = 'J';  name[1] = 'o';  name[2] = 'h';  name[3] = 'n';
name[4] = '\0';
```

Rule: A string in C has to be null terminated

- Example 4:

```
// the null character '/0' is automatically includes by the compiler
 char name[8] = "John"; // notice the double quotation mark " instead of '

// This is INVALID !!
char name[8];
name = "John";
```

- Example 5:

```
// In this case, the null character will NOT be automatically added
char name[5] = {'J', 'o', 'h', 'n', '\0'}
```

- Example 6:

```
// the size of name would be 5 units (5 bytes), thanks to the compiler
char name[] = "John";
```

# character array & pointers 練習

## Example 1:

```
char c1[6] = "Hello";
char *c2;
c2 = c1;
c2[0] = 'A';
*(c1 + 2) = '!';
// c1 = c2;
// c1 = c1 + 1;
c2 ++;
c2[0] = 'A';
```

- What happened in the memory?

- What will be the output if we execute the following code?

```
printf("%s\n", C1);
printf("%s\n", C2);
c2[2] = '\0';
printf("%s\n", C1);
printf("%s\n", C2);
printf("%s\n", C2 + 3);

printf("%c\n", C1[0]);
```

## Example 2:

```
int n = 10;
char c[10] = "UTSC";
int *p = &n;
char *q = &c[0];
*p = *(p+1);
*q = *(q+1);
q = q + 1;
c = c + 1;
```

- In the code above, which line(s) of code may result in an ERROR and Why?

**Another way to create a string**

- As we all know, character array is a character pointer. We can do the following:
  `char c3* = "Hello";`

- IMPORTANT, this string "Hello" is stored in the READ ONLY part in our memory. The following won't work.
  `c3[0] = 'A';`

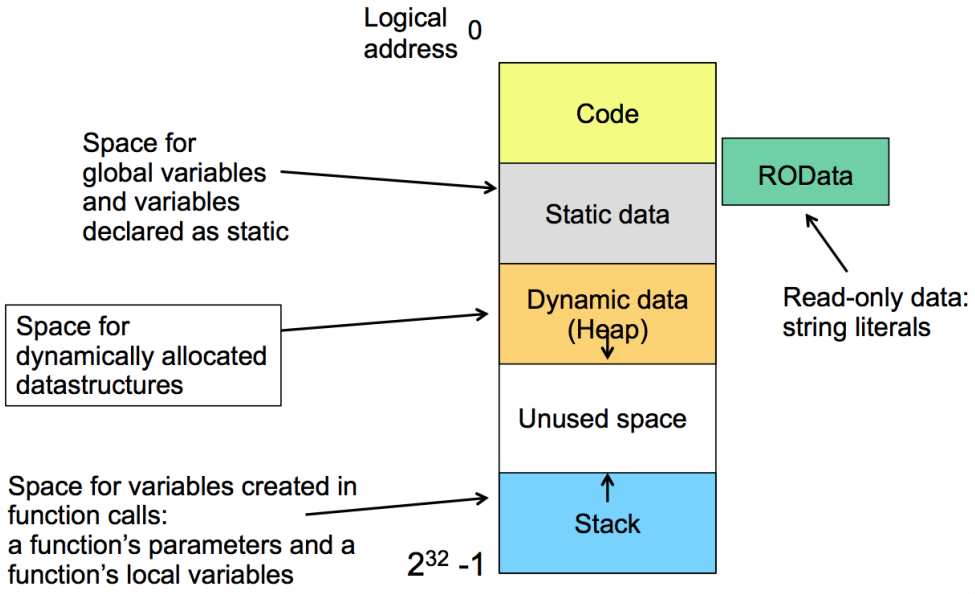
**Yet, another way to create a string**

- We can allocate spaces in the HEAP part of our memory and store the string there.

```
char c3* = (char *) calloc(1, sizeof("Hello"));
char c3* = (char *) calloc(1, strlen("Hello") + 1);
char c3* = (char *) calloc(1, 6);
char c3* = (char *) calloc(6, sizeof(char));
```

- Then, we copy the string into the memory blocks we allocated
  `strcpy(c3, "Hello");`

## Different Types of Strings



## String library

- Library string.h `#include<string.h>`
  - Provide a lot of functions for string manipulation
  - All of the function are assumed that the string would be **null terminated**
  - Useful functions such as:
    - `char *strcpy(char *dest, const char *src);`
    - Copies the string pointed to by src, including the terminating null byte ('\0'), to the buffer pointed to by dest. The strings may not overlap, and the destination string dest must be large enough to receive the copy. The function returns a pointer to the destination string.

    - `char *strcat(char *dest, const char *src);`
    - Appends the src string to the dest string, overwriting the null byte ('\0') at the end of dest, and then adds a terminating null byte. The strings may not overloop, and the dest string must have enough space for the result. The function returns a pointer to the resulting string dest.

    - `size_t strlen(const char *s);`
    - Return the length of the string (number of character in the string, excluding the null terminator).

    - `int strcmp(const char *s1, const char *s2);`
    - Compares the two strings s1 and s2. It returns an integer less than, equal to, or greater than zero if s1 is found respectively, to be less than, to match, or to greater than s2.

# String function examples

```
#include<stdio.h>
#include<string.h>
int main() {
  char name[] = "John";
  printf("Size in bytes = %d\n", sizeof(name));
  int len = strlen(name);
  printf("Length = %d\n", len);

  char s1[30];
  char s2[10];
  // copy a string (initializing a string)
  strcpy(s1, "hello ");
  strcpy(s2, "world");

  printf("%s%s\n", s1, s2);

  // comparing two strings
  int ret;
  ret = strcmp(s1, s2);

  if(ret < 0) {
      printf("str1 is less than str2\n");
  } else if(ret > 0) {
      printf("str2 is less than str1\n");
  } else {
      printf("str1 is equal to str2\n");
  }

  // Concatenating two string
  strcat(s1, s2);
  printf("%s%s\n", s1, s2);

  strcpy(s1, "bye");
  printf("%s\n", s1);

  char s3[10];
  strcat(s3, "vincent");
  printf("%s\n", s3);


  return 0;
}
```

## Pointers 練習

In each subquestion below, fill in the box with the declaration for an appropriate mystery function so that the following code would compile without error. Each subquestion is independent.

```
char ** students;
int class_size = 100;
if(mystery1(class_size, &students) > 0)
    printf("There were errors\n");
```

```
char ** authors;
char * most_famous;
int books[4] = {4,12,16,22};
int * p = books;

most_famous = mystery2(books[0], &p, *authors);
```

```
int rating[3] = {4, 5, 10};
char ** items;
char name[30];
strcpy(name, mystery3(rating[2], &items));
```

Consider the following C code.

```
char data[128] = "How much wood could a woodchuck chuck?";
```

We want to use a variable `ptrs` to point to the first character of each word in `data`. Use `calloc` so that enough memory is allocated to store a pointer to the first character of each word in `data`.

Using array notation, write a statement that makes the first element in `ptrs` refer to the first character of the first word in `data`.

Using only pointer notation, write a statement that makes the second element in `ptrs` refer to the first character of the second word in data without changing the value of `ptrs`.

Write two C statements so that the following statement prints “wood”.

```
printf("%s\n", *ptrs);
```

In this question, you will implement your own version of `strlen`. Your `strlen` should have the same interfacae and functionality as the corresponding C library function. You are not allowed to use any C library functions in your implementation. Fill in the body of the function below:

```
int strlen(const char *s) {
```

Implement your own version of `strcat`. Your `strlen` should have the same interfacae and functionality as the corresponding C library function. You are not allowed to use any C library functions in your implementation, but you can call your `strlen` function you implemented above. Fill in the body of the function below:

```
char *strcat(char *dest, const char *src) {
```