Week 4- lecture 1, 2

Pointers

Edited by: Dr. Saeid Pourroostaei Ardakani

Autumn 2021

https://www.vectorstock.com/royalty-free-vector/smiling-donkey-head-vector-1217480
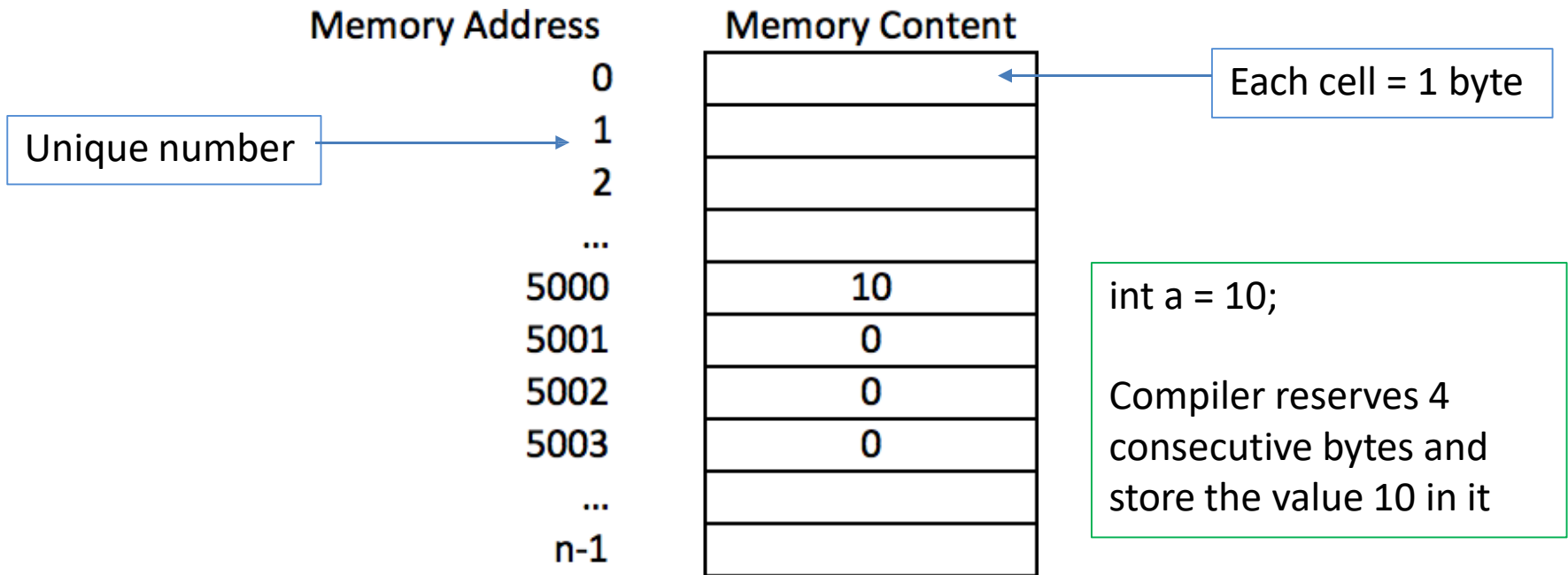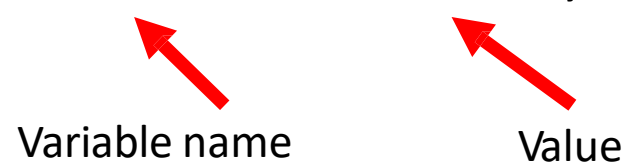
Overview

- Declaration and initialisation

- Pointer to Constant vs. const Pointer

- Pointers and arrays

  - String literals

- Array of pointers

- Pointer arithmetic (e.g. subtracting, comparing)

Memory Layout

Memory Address	Memory Content
0	
1	
2	
...	
5000	10
5001	0
5002	0
5003	0
...	
n-1	

Unique number

Each cell = 1 byte

int a = 10;

Compiler reserves 4 consecutive bytes and store the value 10 in it

University of Nottingham
UK | CHINA | MALAYSIA

Var Name, Val and Mem Address

- int ID = 2017233;

Variable name   Value

- &ID

Memory address of ID

Analogy:

StudentA received a 1st class degree.

StudentA lives in Building #23.

```
C:\Users\z2017233\Desktop>iteration
Current ID number is 0
Current ID number is 0060FF2C

Enter your ID number: 2017233

Current ID number is 2017233
Current ID number is 0060FF2C

C:\Users\z2017233\Desktop>
```

```c
#include <stdio.h>

int main(void)
{
    int id = 0;

    printf("Current ID number is %d\n", id);
    printf("Current ID number is %p\n", &id);

    printf("\n\nEnter your ID number: ");
    scanf("%d", &id);

    printf("\n\nCurrent ID number is %d\n", id);
    printf("Current ID number is %p\n", &id);

    return 0;
}
```

Pointer and Variable

- Pointers are variables whose values are memory addresses.

- Pointers enable programs to:

    - simulate pass-by-reference

    - pass functions between functions

    - create and manipulate dynamic data structures, i.e., data structures that can grow and shrink at execution time, such as linked lists, queues, stacks and trees.
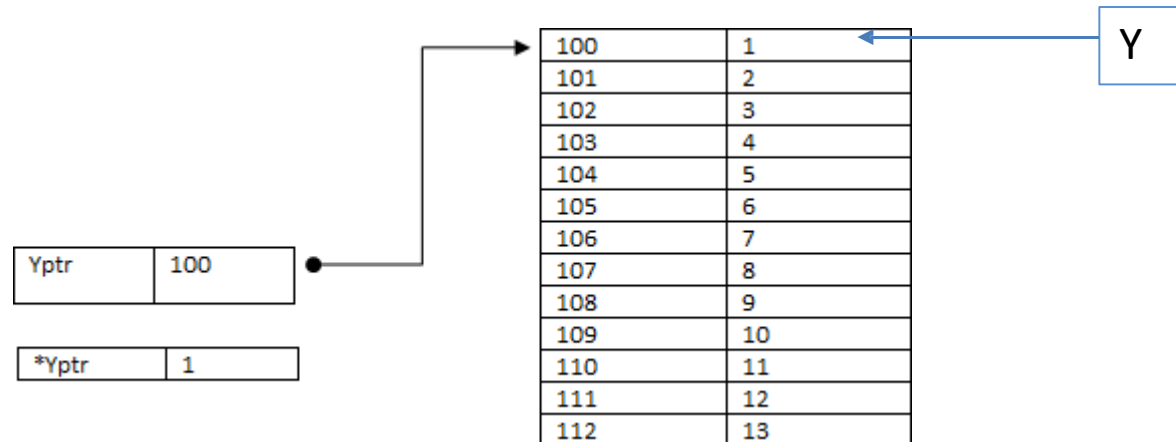
Pointer and Variable (2)

```
// normal integer initialised to value 5
int x = 5;
// declare a pointer to an integer variable
int *px;
// set the pointer value to the address of the
x variable
px = &x;
```

Pointer and Variable (3)

- A variable name *directly* references a value, a pointer ***indirectly*** references a value.

```
int Y = 1;
int *Yptr;
Yptr = &Y;
```

Yptr	100

*Yptr	1

100	1
101	2
102	3
103	4
104	5
105	6
106	7
107	8
108	9
109	10
110	11
111	12
112	13

Y

Source: http://www.exforsys.com/tutorials/c-language/c-pointers.html

Pointer and Variable (4)

- A pointer may be initialized to NULL, 0 or an address.

- A pointer with the value NULL points to nothing.

```
int *px = NULL;
// ...
// do some things that may or may not
// make px point to a variable.
// ...
if(px != NULL)
{
    printf("%d\n", *px);
}
```

Example: a simple pointer

```c
1   // Fig. 7.4: fig07_04.c
2   // Using the & and * pointer operators.
3   #include <stdio.h>
4
5   int main( void )
6   {
7      int a; // a is an integer
8      int *aPtr; // aPtr is a pointer to an integer
9
10     a = 7;
11     aPtr = &a; // set aPtr to the address of a
12
13     printf( "The address of a is %p"
14             "\nThe value of aPtr is %p", &a, aPtr );
15
16     printf( "\n\nThe value of a is %d"
17             "\nThe value of *aPtr is %d", a, *aPtr );
18
19     printf( "\n\nShowing that * and & are complements of "
20             "each other\n&*aPtr = %p"
21             "\n*&aPtr = %p\n", &*aPtr, *&aPtr );
22  } // end main
```

Example (output)

```
The address of a is 0028FEC0
The value of aPtr is 0028FEC0

The value of a is 7
The value of *aPtr is 7

Showing that * and & are complements of each other
&*aPtr = 0028FEC0
*&aPtr = 0028FEC0
```

Declaring Pointers

- Pointers hold **memory address** of another variable
- **int** *int_ptr, myInt;
- **double** *double_ptr, myDouble;

- sizeof(int_ptr) == sizeof(double_ptr) == sizeof(myInt) = 4 bytes
- sizeof(myDouble) = 8 bytes

University of Nottingham
UK | CHINA | MALAYSIA

Pointer Initialisation

- Memory address operator is &

- int *ptr;
  int a = 0;
  ptr = &a;

Careful!!, If pointer is used without initialisation, it can cause segmentation fault

- int *ptr = NULL;

Pointer that **does NOT** point to anything.

University of Nottingham
UK | CHINA | MALAYSIA

Example: Pointer and Variable

- int num = 50;

- int *ptr = #

- **Variable**   **Value in it**
  num   50
  &num   1002
  ptr   1002
  *ptr   50

Variable Name : **num**

50

1002

Source: http://www.c4learn.com/c-programming/c-dereferencing-pointer/

University of
Nottingham
UK | CHINA | MALAYSIA

Example: Pointer and Variable (2)

- Memory addresses are unchanged.

- Values can be changed.
    - For pointer, the change of value means the change of location (where it is pointing to).

De-referencing of Pointer

- To read the value at a given memory address

```
99      int main(void)
100   {
101         int x = 5;
102         int *p = NULL;
103
104         p = &x;
105
106         printf("%d\n", *p); // dereference
107
108         printf("%p\n", &p);
109         printf("%p\n", p);
110         printf("%p\n", &x);
111
112         // NOTE: the output when printing out p and &x is the same because p is a pointer
113         //        and it is pointing to x, therefore memory address of x is sotred in p
114
115         return 0;
116   }
```
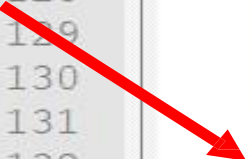
University of Nottingham
UK | CHINA | MALAYSIA

De-referencing of Pointer (2)

- To write the value at a given memory address

```
121    int main(void)
122  ⊟{
123        int x = 5;
124        int *p = NULL;
125
126        p = &x;
127
128        printf("%d\n", *p); // dereference
129        printf("%p\n", p);
130        printf("x is %d\n", x);
131
132        *p = 7; // dereference
133        printf("\n%d\n", *p); // dereference
134        printf("%p\n", p);
135        printf("x is %d\n", x);
136
137
138        return 0;
139  └}
```

University of Nottingham
UK | CHINA | MALAYSIA

Suggestions – Code Spacing

- Compile error

Example: good spacing

```
        *average = *total / *count;    /* compute the average */
```

Example: poor spacing

```
        *average=*total/*count;         /* compute the average */
                        ^  begin comment  end comment^
```

University of Nottingham

UK | CHINA | MALAYSIA

Q1: What is the output?

- int *ptr, a;
  a = 10;
  ptr = &a;
  printf("Val = %d\n", *ptr);

  a) 5000
  b) 10
  c) 5003
  d) 5010

Memory Address	Memory Content
0	
1	
2	
...	
5000	10
5001	0
5002	0
5003	0
...	
n-1	

Q2: What is the output?

```
int *pc, c;
c = 5;
pc = &c;
c = 1;
printf("%d", c);
printf("%d", *pc);
```

a) 5, 1
b) 1, 1
c) 5, 5
d) 1, 5

Q3: What is the output?

```
int *pc, c, d;
c = 5;
d = -15;
pc = &c;
printf("%d", *pc);
pc = &d;
printf("%d", *pc);
```

a) 5, -15
b) -15, 5
c) 5, 5
d) -15, -15

Q4: What is the output?

```c
#include <stdio.h>
int main()
{
  int *pc, c;

  c = 22;
  printf("Address of c: %p\n", &c);
  printf("Value of c: %d\n\n", c);

  pc = &c;
  printf("Address of pointer pc: %p\n", pc);
  printf("Content of pointer pc: %d\n\n", *pc);

  c = 11;
  printf("Address of pointer pc: %p\n", pc);
  printf("Content of pointer pc: %d\n\n", *pc);

  *pc = 2;
  printf("Address of c: %p\n", &c);
  printf("Value of c: %d\n\n", c);
  return 0;
}
```

Q5: what is the output?

```
int i = 0, *ptr = &i;
*ptr = *ptr ? 10 : 20;
printf("Val = %d\n", i);
```

a) 10
b) 0
c) 455839228 (a mem addr)
d) 20

Q6: what is the output?

- int *ptr1, *ptr2, *ptr3, i = 10, j = 20, k = 30;
  ptr1 = &i;
  ptr2 = &j;
  ptr3 = &k;
  *ptr1 = *ptr2 = *ptr3;
  k = i+j;
  printf("%d\n", *ptr3);

a) 20
b) 30
c) 50
d) 60

Q7: what is the output?

- int *ptr1, *ptr2, *ptr3, i = 10, j = 20, k = 30;
  ptr1 = &i;
  i = 100;
  ptr2 = &j;
  j = *ptr2 + *ptr1;
  ptr3 = &k;
  k = *ptr3 + *ptr2;
  printf("%d %d %d\n", *ptr1, *ptr2, *ptr3);

a) 100, 120, 150
b) 100, 150, 120
c) 110, 110, 130
d) 110, 130, 110

Overview

- Declaration and initialisation
- **Pointer to Constant vs. const Pointer**
- Pointers and arrays
  - String literals
- Array of pointers
- Pointer arithmetic (e.g. subtracting, comparing)
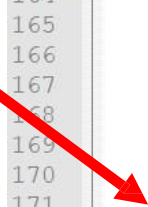
Pointer to a const Variable

- **A non-constant pointer to constant data can be modified to point to any data item of the appropriate type, but the data to which it points cannot be modified.**

  - Such a pointer might be used to receive an array argument to a function that will process each element without modifying the data.

Pointer to a const Variable (2)

- Here, pointer can be used to change the data.
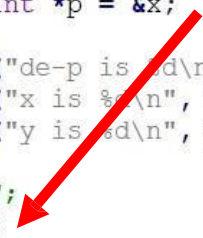
```
160    #include <stdio.h>
161
162    int main(void)
163    {
164        const int x = 5, y = 6;
165        int *p = &x;
166
167        printf("de-p is %d\n", *p); // dereference
168        printf("x is %d\n", x);
169        printf("y is %d\n", y);
170
171        //x = 7;
172        *p = 7;
173        printf("\n\nde-p is %d\n", *p); // dereference
174        printf("x is %d\n", x);
175        printf("y is %d\n", y);
176
177        p = &y;
178        printf("\n\nde-p is %d\n", *p); // dereference
179        printf("x is %d\n", x);
180        printf("y is %d\n", y);
181
182
183        return 0;
184    }
```

"Pointer to Constant" to const Var

- Here, **compile error**!!

- Pointer to constant can NOT be used to change data.

```c
#include <stdio.h>

int main(void)
{
    const int x = 5, y = 6;
    const int *p = &x;

    printf("de-p is %d\n", *p); // dereference
    printf("x is %d\n", x);
    printf("y is %d\n", y);

    //x = 7;
    *p = 7;
    printf("\n\nde-p is %d\n", *p); // dereference
    printf("x is %d\n", x);
    printf("y is %d\n", y);

    p = &y;
    printf("\n\nde-p is %d\n", *p); // dereference
    printf("x is %d\n", x);
    printf("y is %d\n", y);


    return 0;
}
```

University of Nottingham
UK | CHINA | MALAYSIA

Constant Pointer to non-constant Var

- A constant pointer to non-constant data always points to the same memory location, and the data at that location can be modified through the pointer.

- Pointers that are declared "const" must be initialized when they're defined.

const Pointer to non-constant Var (2)

- Prohibits a pointer from changing **the value of the variable** it points to

- int j, i = 10;
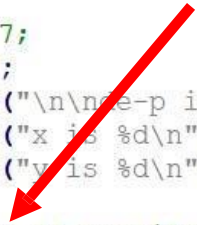 const int *ptr;
  ptr = &i;

- *ptr = 30; ←——— Not allowed: the programme won't compile

const Pointer to non-constant Var (3)

- Here, **compile error**!!

- const Pointer can NOT change where it is pointed to.

```c
241    #include <stdio.h>
242
243    int main(void)
244  ⊟{
245        int x = 5, y = 6;
246        int *const p = &x;
247
248        printf("de-p is %d\n", *p); // dereference
249        printf("x is %d\n", x);
250        printf("y is %d\n", y);
251
252        //x = 7;
253        *p = 7;
254        printf("\n\nde-p is %d\n", *p); // dereference
255        printf("x is %d\n", x);
256        printf("y is %d\n", y);
257
258        p = &y; // compile error
259        printf("\n\nde-p is %d\n", *p); // dereference
260        printf("x is %d\n", x);
261        printf("y is %d\n", y);
262
263
264        return 0;
265  └}
```

Pointer to Constant vs. const Pointer

- Pointer to Constant
const int* ptr = &x;

  const Pointer
int *const ptr = &x;

  Variable it is pointed to:
can be modified
e.g. ptr = &y;

  CANNOT

  Value pointed by the pointer:
CANNOT

  can be modified
e.g. *ptr = 7;

Constant Pointer to constant Var

- The least access privilege is granted by a constant pointer to constant data.

- Such a pointer always points to the same memory location, and the data at that memory location cannot be modified.

Constant Pointer to constant Var (2)

- Prohibits a pointer from **pointing to another variable**

- const int j=20, i = 10;
  int *const ptr = &i;
  *ptr = 30;

- ptr = &j;

Overview

- Declaration and initialisation
- Pointer to Constant vs. const Pointer
- **Pointers and arrays**
  - **String literals**
- Array of pointers
- Pointer arithmetic (e.g. subtracting, comparing)

Pointers and Arrays

- The elements of an array are stored in successive memory location

- int arr[2];

- The first element is stored in 5000 – 5003

- The second element is stored in 5004 – 5007

- arr == &arr[0]

Memory Address	Memory Content
0	
1	
2	
...	
5000	10
5001	0
5002	0
5003	0
...	
n-1	

Name of an array can be used as a pointer to its first element!!

University of Nottingham
UK | CHINA | MALAYSIA

Use Pointer Variable like Array

- int *ptr, i, arr[5] = {10, 20, 30, 40, 50};
  ptr = arr;   Points to the first element
  for(i = 0; i < 5; i++){
      printf("Addr = %p Val = %d\n", ptr, *ptr);
      ptr++;   Increment the pointer by 4 bytes
  }   **WHY?!**

Address and value

Pointer to Array

- How many of the printfs below have the same output??

```
299    int x[] = {9, 11, 13};
300    int *p;
301    p = x; // array name is a pointer, and pointer stores memory address!!
302
303    int y = 10;
304    int *q;
305    q = &y; // note the difference when pointer is pointing to an array
306           // and when pointer is pointing to a normal variable
307
308    printf("%p\n", x);
309    printf("%p\n", &x[0]);
310    printf("%p\n", p);
311    printf("%p\n", &p);
```
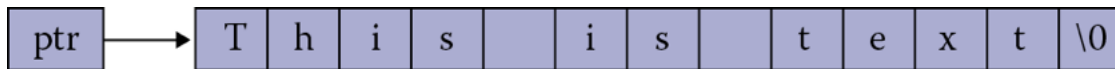
```
19035@CSLinux PGA-w4l1]$ ./ex7
fe7e971dd0
fe7e971dd0
fe7e971dd0
fe7e971dc8
19035@CSLinux PGA-w4l1]$
```

University of Nottingham
UK | CHINA | MALAYSIA

Pointers and String Literals

- printf("%c %c\n", "message"[4], *("message"+4));

- char *ptr = "This is text";

String literals is usually read-only, so you might not be able to modify its content

- printf("%c %c\n", ptr[5], *(ptr+5));
  printf("%s\n", ptr);
  printf("%s\n", ptr+5);

Example: Your own strlen

- Relies on having '\0' to terminate your string.
- Otherwise, undefined behaviour.

```
435    #include <stdio.h>
436
437    int main(void)
438    {
439        // Can declare your string as char *
440        // The const keyword enforces the fact you are not allowed to change
441        // a string literal.
442        // const char *str = "Paul";
443
444        // Or you could declare the string as an array of characters.
445        // Try changing the nul character at the end to something else to see
446        // what happens with string not ending in \0.
447        char name[] = {'P','a','u','l','\0'};
448        char *str = name;   // "name" equivalent to "&name[0]"
449
450        // First version that uses array notation.
451        int n = 0;
452        while(str[n] != '\0')
453        {
454            //printf("%c", str[n]);   // Comment out to print characters 1 by 1
455                                       // while counting.
456            n = n + 1;
457        }
458        printf("\nArray notation length %d.\n", n);
459
460        // Second version that uses pointer arithmetic.
461        int len = 0;
462        while(*str != '\0')
463        {
464            //printf("%c", *str);   // Comment out to print characters 1 by 1
465                                     // while counting.
466            str = str + 1;
467            len = len + 1;
468        }
469        printf("\nPointer arithmetic length %d.\n", len);
470
471        return 0;
472    }
```

University of Nottingham
UK | CHINA | MALAYSIA

String Functions

- #include <string.h>
- strlen() – not counting null character
- strcpy(*dest, *src)   Check if dest is big enough!!
- strncpy(*dest, *src, count)   Add null character if src is shorter than count
- strcat(*dest, *src)
- strcmp(*dest, *src)   Negative for less or shorter, positive for more, zero for identical
- strncmp(*dest, *src, count)

Read more here: https://beginnersbook.com/2014/01/c-strings-string-functions/

University of Nottingham
UK | CHINA | MALAYSIA

Q8: What will be displayed?

- int *ptr, arr[5] = {10, 20, 30, 40, 50}; ptr = arr;

  printf("Val1 = %d, Val2 = %d\n", *ptr+2,*(ptr+2));

    a) Val1 = 22, Val2= 20
    b) Val1 = 22, Val2= 30
    c) Val1 = 12, Val2= 20
    d) Val1 = 12, Val2= 30

Q9: What is arr[0] + arr[2]?

- int *ptr, arr[] = {10, 20, 30, 40, 50};
  ptr = arr;
  *ptr = 3;
  ptr += 2;
  *ptr = 5;
  printf("Val = %d\n", arr[0]+arr[2]);

  a) 10
  b) 30
  c) 40
  d) 8

Summary

- Declaration and initialisation
- Pointer to Constant vs. const Pointer
- Pointers and arrays
  - String literals