Basic and Practice in Programming
Lab7
Variable and Its Address (1/2)

• What is the variable?
  – Abstracted representation of allocated memory
  – Having address & value

```
int a = 10;
```

![Diagram showing memory allocation and addressing]
/* Practice 1 : Variable and Address */
#include <stdio.h>

int main(void)
{
    int a = 10;

    printf("value: %d address: %p\n", a, &a);

    return 0;
}
A Consideration

• Let’s consider follow function form

```c
int add (int x, int y)
{
    x += y;
    return x;
}
```
A Consideration

• Let’s consider follow function form

```c
int add (int x, int y)
{
    x += y;
    return x;
}
```

Realistic executable function form

```c
int add (_x, _y)
{
    int x, y;
    x = _x;
    y = _y;
    x += y;
    return x;
}
```

The function uses arguments as form of local variables
A Consideration

- Let’s consider follow function form

```c
int add (int x, int y)
{
    x += y;
    return x;
}
```

```c
int main (void)
{
    int x = 10, y = 10, sum;
    sum = add(x, y);
    return 0;
}
```

**Realistic executable function form**

```c
int add (_x, _y)
{
    int x, y;
    x = _x;
    y = _y;
    x += y;
    return x;
}
```

**What is the value of x?**

The function uses arguments as form of local variables
/* Practice 2 : Call by value */
#include <stdio.h>

int add (int x, int y)
{
    x += y;
    return x;
}

int main(void)
{
    printf("%d + %d = %d\n", 10, 20, add(10, 20));

    return 0;
}
/* Practice 3 : Call by value 2 */
#include <stdio.h>

int add (int x, int y)
{
    x += y;
    printf("x: %p y: %p", &x, &y);
    return x;
}

int main(void)
{
    int x = 10, y = 20;
    printf("x: %p y: %p", &x, &y);
    printf("%d + %d = %d\n", x, y, add(x, y));
    return 0;
}
A Question

• How to use same “variable”
  – Across two functions

• The answers are
  – Using global variable but,
    • Using global variable is NOT recommended
    • Security, consistency, reliability problems
  – Using variable’s address instead of variable itself
    • Pointer
Pointer

• It is not a concept
  – Just a method
  – There is no serious “philosophy” in the pointer
  – Pointer is just variable for containing address

```c
int a = 10;
int *p = &a;  //declaration form of pointer

//p == &a, *p == a
```
int a = 10;
int *p;
*p = &a;


Functions

• Two types of function call
  – Call by value
  – Call by reference

• Call by value
  – Argument is just “value”

• Call by reference
  – Argument is given by “variable’s address”

```c
void func1(int a) {
    ...
}

void func2(int *a) {
    ...
}
```
Call by Reference (1/3)

- **Swap function**
  - Exchange two variable’s contents

```c
int a = 10, b = 20;
swap(a, b);
```

Before swap:

```
a = 10
b = 20
```

After swap:

```
a = 20
b = 10
```
/* Practice 4 :
   Call by reference example1*/
#include <stdio.h>

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

int main(void)
{
   int a = 10, b = 20;

   printf("a : %d b : %d", a, b);
   swap(a, b);
   printf("a : %d b : %d", a, b);

   return 0;
}
/* Practice 5: 
Call by reference example2*/

#include <stdio.h>

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

int main(void)
{
    int a = 10, b = 20;

    printf("a : %d b : %d", a, b);
    swap(&a, &b);
    printf("a : %d b : %d", a, b);

    return 0;
}
Reference?

Call by value

main

\[
\begin{array}{c|c}
\text{a} & 10 \\
0x1000 & \\
\end{array}
\]

\[
\begin{array}{c|c}
\text{b} & 20 \\
0x1004 & \\
\end{array}
\]

swap

\[
\begin{array}{c|c}
\text{x} & 10 \\
\text{y} & 20 \\
\end{array}
\]
Reference?

Call by value

main

<table>
<thead>
<tr>
<th>a</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0x1000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>b</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0x1004</td>
</tr>
</tbody>
</table>

call

swap

<table>
<thead>
<tr>
<th>x</th>
<th>20</th>
</tr>
</thead>
</table>

| y  | 10 |

0x1000
0x1004
**Reference?**

**Call by value**

```
main
a  10
  0x1000
b  20
  0x1004
```

```
swap
x  20
y  10
```

**Call by reference**

```
main
a  10
  0x1000
b  20
  0x1004
```

```
swap
*x  0x1000
*y  0x1004
```
Reference?

Call by value

Call by reference

main

a
10
0x1000

b
20
0x1004

swap

x
20

y
10

swap

*x
0x1000

*y
0x1004

0x1000
10

0x1004
20
Reference?

Call by value:
- `main` assigns `a` to `20` at `0x1000` and `b` to `10` at `0x1004`.
- `swap` receives `x` at `0x1000` and `y` at `0x1004`.

Call by reference:
- `main` assigns `a` to `20` at `0x1000` and `b` to `10` at `0x1004`.
- `swap` updates `*x` at `0x1000` to `20` and `*y` at `0x1004` to `10`.

Reference variables are used to refer to variables by address, allowing direct manipulation of the values they point to.
Type

• Pointer has ‘type’
  – Integer, character, floating point…
  – Pointer must be used to point to same type variable

```
int *ip
ip = &int_val
int int_val = 10;

char *cp
cp = &char_val
char char_val = ‘a’;

float *fp
fp = &f_val
float f_val = 3.14;
```
• Pointer is just a variable
  – Of course, it has its address

/* Practice 4 :
   Pointer of pointer */
#include <stdio.h>

int main (void)
{
  int x = 10, *p, **pp;
  p = &x;
  pp = &p;
  printf ("%d %d %d\n", x, *p, **pp);
  printf ("%p %p %p\n", &x, p, *pp);
  printf ("%p %p\n", &p, pp);
  return 0;
}
Exercise 1

• Left shifter
  – Input is given as form of sequence of 5 distinct characters
  – Last input is “the number of shift”
  – Skeleton code is given
    • Check the homepage
  – You must not print last input

A B C D E 3 → D E A B C
Array (1/8)

• Pointer
  – Container of memory address
  – Accessing variable’s value by using ‘*’

```
int *p;
p = &a;
```

• Array
  – Reference of continuous memory addresses
  – Not a variable

```
int arr[10]
arr == &arr[0]
*arr?
```
/ * Practice 1 : Array 1*/
#include <stdio.h>

int main (void)
{
    int arr[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

    printf("%p %p\n", &arr[0], arr);
    printf("%d %d\n", arr[0], *arr);

    return 0;
}
Array (3/8)

- Array in memory
  - Address is linearly increasing
    - By size of type

```
int arr[5]
```

<table>
<thead>
<tr>
<th>Index</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>arr[0]</td>
<td>0x1000</td>
</tr>
<tr>
<td>arr[1]</td>
<td>0x1004</td>
</tr>
<tr>
<td>arr[2]</td>
<td>0x1008</td>
</tr>
<tr>
<td>arr[3]</td>
<td>0x100c</td>
</tr>
<tr>
<td>arr[4]</td>
<td>0x1010</td>
</tr>
</tbody>
</table>

sizeof(int)
/* Practice 2 : Array 2*/
#include <stdio.h>

int main (void)
{
    int arr[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    int i;

    for (i=0;i<10;i++)
        printf("%p %p\n", &arr[i], arr+i);

    for (i=0;i<10;i++)
        printf("%d %d\n", arr[i], *(arr+i));

    return 0;
}
/* Practice 3 : Array 3*/
#include <stdio.h>

int main (void)
{
    char arr[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    int i;

    for (i=0;i<10;i++)
        printf("%p %p\n", &arr[i], arr+i);

    for (i=0;i<10;i++)
        printf("%d %d\n", arr[i], *(arr+i));

    return 0;
}
Array (6/8)

- 2D array

```
int arr[2][5]
```

```
<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>arr[0][0]</td>
<td>0x1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>arr[0][1]</td>
<td>0x1004</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>arr[0][2]</td>
<td>0x1008</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>arr[0][3]</td>
<td>0x100c</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>arr[0][4]</td>
<td>0x1010</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>arr[1][0]</td>
<td>0x1014</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>arr[1][1]</td>
<td>0x1018</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

- `sizeof(int)`
- `sizeof(int) * 5`
- `sizeof(int)`
/* Practice 4 : 2D Array 1*/
#include <stdio.h>

int main (void)
{
    int arr[2][5] = {{0, 1, 2, 3, 4}, {5, 6, 7, 8, 9}};

    printf("%p %p %p\n", arr, arr[0], &arr[0][0]);
    printf("%p %p %p\n", arr+1, arr[1], &arr[1][0]);
    printf("%p %p %p\n", *(arr+1)+2, arr[1]+2, &arr[1][2]);

    return 0;
}
/* Practice 5 : 2D Array 2*/
#include <stdio.h>

int main (void)
{
    int arr[2][5] = {{0, 1, 2, 3, 4}, {5, 6, 7, 8, 9}};

    printf("%p %p %p\n", arr, arr[0], &arr[0][0]);
    printf("%p %p %p\n", arr+1, arr[1], &arr[1][0]);
    printf("%p %p %p\n", *(arr+1)+2, arr[1]+2, &arr[1][2]);

    printf("%d %d %d\n", **arr, *arr[0], arr[0][0]);
    printf("%d %d %d\n", **(arr+1), *arr[1], arr[1][0]);
    printf("%d %d %d\n", *(*(arr+1)+2), *(arr[1]+2), arr[1][2]);

    return 0;
}
• Pointer can be used as array

```c
int arr[5];
int *p;
p = &arr[0]; //same as p = arr;
```
/* Practice 6: Pointer 1*/
#include <stdio.h>

int main (void)
{
    int arr[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    int i;
    int *p;
    p = &arr[0]; //same as p = arr

    for (i=0;i<10;i++)
        printf("%p %p %p\n", &p[i], p+i, &arr[i]);

    for (i=0;i<10;i++)
        printf("%d %d %d\n", p[i], *(p+i), arr[i]);

    return 0;
}
Pointer must be carefully used

```c
int arr[5];
int *p;
printf("%d\n", p[1]);
```
/ * Practice 7 : Pointer misuse*/
#include <stdio.h>

int main (void)
{
    int arr[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    int i;
    int *p;
    //p = &arr[0];

    for (i=0;i<10;i++)
        printf("%p %p %p\n", &p[i], p+i, &arr[i]);

    for (i=0;i<10;i++)
        printf("%d %d %d\n", p[i], *(p+i), arr[i]);

    return 0;
}
/* Practice 8 : Pointer 2*/
#include <stdio.h>

int string_length(char *s) {
    int n;
    for (n = 0; *s != '\0'; s++)
        n++;
    return n;
}

int main (void)
{
    char str[10];
    scanf("%s", &str[0]);
    printf("%s\nLength : %d\n", str, string_length(str));
    return 0;
}
• String copy
  – “strcpy()” is defined in standard C library
/* Practice 9 : String copy 2*/
#include <stdio.h>

void mystrcpy(char s[], char d[]) {
    int i;
    for (i = 0; s[i] != '\0'; i++)
        d[i] = s[i];
    d[i] = s[i];
}

int main (void) {
    char str[10], dest[10];
    scanf("%s", &str[0]);
    printf("%s\nLength : %d\n", str, string_length(str));
    return 0;
}
Pointer Array

- Pointer can be declared as array
  - Example: string array

```c
char *str_array[5]
```

- `str_array[0]` points to “All work”
- `str_array[1]` points to “and”
- `str_array[2]` points to “no play”
- `str_array[3]` points to “makes”
- `str_array[4]` points to “Jack a dull boy”
/* Practice 10 : Pointer array */
int main (void)
{
    char *str_array[5] = {"All work", "and", "no play",
                           "makes", "Jack a dull boy"};

    printf("%s %s %s %s %s\n", str_array[0], str_array[1],
            str_array[2], str_array[3], str_array[4]);

    return 0;
}
/* Practice 11 : Pointer array 2 */
int main (void)
{
    char *str_array[5] = {“All work”, “and”, “no play”,
                          “makes”, “Jack a dull boy”};

    printf(“%s %s %s %s %s\n”, str_array[0], str_array[1],
            str_array[2], str_array[3], str_array[4]);
    str_swap(&str_array[0], &str_array[2]);
    printf(“%s %s %s %s %s\n”, str_array[0], str_array[1],
            str_array[2], str_array[3], str_array[4]);

    return 0;
}

void str_swap (char **s, char **d)
{
    char *temp;
    temp = *s;
    *s = *d;
    *d = *s;
}
Exercise 1

• Modifying strcpy()
  – Two arguments
    • Source string
    • Destination string
  – Don’t use local variable
    • Using pointer operations
  – Modifying practice 9’s code
Exercise 2

- Word changer
  - Input is composed of three strings
    - A sentence not including white space
    - Target word
    - Changing word
  - You find target words in a sentence, and change by changing word
    - Two words have same length

All work and no play makes Jack a dull boy

All play and no play makes Jack a dull boy