Basic and Practice in Programming
Lab7
Variable and Its Address (1/3)

```
int a = 10;
```

variable value

How to understand this statement at machine’s perspective?
• What is the variable?
  – Abstracted representation of allocated memory
  – Having address & value

```
int a = 10;
```

```
<table>
<thead>
<tr>
<th>address</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000010</td>
<td>10</td>
</tr>
<tr>
<td>0x00000004</td>
<td></td>
</tr>
<tr>
<td>0x00000000</td>
<td></td>
</tr>
</tbody>
</table>
```

```
0x00000010 ← -- &a
```
/* 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 add (_x, _y)
{
    int x, y;
    x = _x;
    y = _y;
    x += y;
    return x;
}
```

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

What is the value of x?

Realistic executable function form

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

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

//p == &a, *p == a
```c
int a = 10;
int *p;
p = &a;
```
• 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”
• Swap function
  – Exchange two variable’s contents

```
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;
}
Call by value

main

\[
\begin{array}{c}
\text{a} \\
10 \\
0x1000 \\
\text{b} \\
20 \\
0x1004 \\
\end{array}
\]

swap

\[
\begin{array}{c}
x \\
10 \\
\hline \\
y \\
20 \\
\end{array}
\]
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>

swap

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

| y  | 10 |

0x1000
0x1004
Reference?

Call by value

Call by reference

main

\[
\begin{array}{c}
\text{a} \\
10 \\
0x1000 \\
\text{b} \\
20 \\
0x1004 \\
\end{array}
\]

swap

\[
\begin{array}{c}
x \\
20 \\
\text{y} \\
10 \\
\end{array}
\]

\[
\begin{array}{c}
*x \\
0x1000 \\
*y \\
0x1004 \\
\end{array}
\]
Call by value

Call by reference

Reference?
Reference?

Call by value

Call by reference

main

a

20

0x1000

b

10

0x1004

swap

x

20

y

10

0x1000

0x1004

Reference

0x1000

20

0x1004

10

swap

*x

0x1000

*y

0x1004
**Type**

- Pointer has ‘type’
  - Integer, character, floating point...
  - Pointer must be used to point to same type variable

```plaintext
int *ip
char *cp
float *fp
```

- `ip = &int_val`
- `cp = &char_val`
- `fp = &f_val`

```plaintext
int int_val = 10;
char char_val = 'a';
float f_val = 3.14;
```
• Pointer is just a variable
  – Of course, it has its address

```c
/* Practice 5 : 
   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