CS1100 - Lecture 6

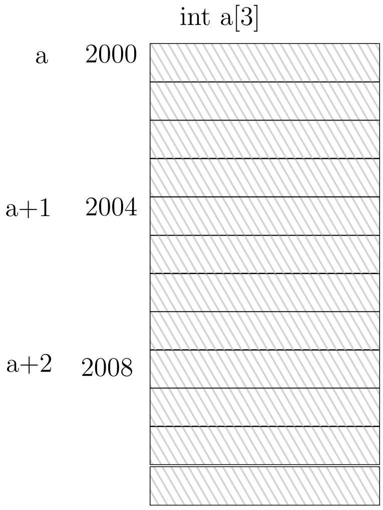
Instructor : Jasine Babu

Teaching Assistants : Nikhila K N, Veena Prabhakaran

So far, we were concerned more about variables and their values. There is an address to which each variable is bound to. Now we will look at more explicit use of addresses.

Before getting into that topic specifically, we will discuss a little bit more about the name used for representing an array. In the declaration `int a[100];`, the label `a` represents 100 integer variables together. It also represents an address, which is the starting address of the first element in the array. But `a` itself is not a variable. We cannot assign a value to the label `a` (i.e., `a=5` or `a=10` is invalid). But we can use the label `a` in certain expressions like `a+1` and `a+5`, which are valid. The expression `a+5` represents the address of the element `a[5]`. When `counter` is an integer variable, `a+counter` is also a valid expression, which represents the the address of the element `a[counter]`.

Now, there is one clarification needed. So far, for simplicity we were assuming that all variables take only one unit of memory. But in reality, each element in array might take more space, not just one location. For example, an integer, in reality, usually takes 4 locations instead of one location. This is because, the number of digits to represent an integer value requires more than one location in practical. Actually the value of `a[0]` is not just stored at the address 2000, rather it may be spread in the locations 2000 to 2003 as shown in the next figure.



In this case, `a+1` actually represents the address 2004 and not 2001. In general, when `a` is an array of integers, `a+i` represents `starting address of the first element of a + (i × number of locations required to store an integer)`. As a programmer, we don't need to worry much about the actual value of this address. For us, `a+i` is always the address of the element $a[i]$. This interpretation does not change, even if the number of locations required to store an integer changes from 4 to 8.

Pointers

Pointers are the special types of variables permitted in C. The declaration `int *p` means, `p` is a variable, which is a pointer to an integer variable. i.e. the value of `p` is the address of some other integer variable. Note that, since `p` is also a variable, it has got an address of its own. Just like usual variables, we can also have many pointer variables in a program.

We know that, pointers are the variables, that can hold addresses of another variables. If a pointer `p` is declared as `int *p`, `p` can hold address of an integer variable.

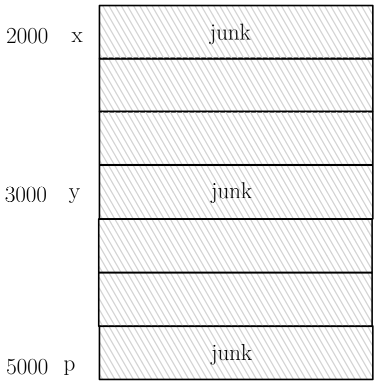
There are two major operations using pointers. The first operation is *address of* operation which is represented using the symbol `&`. If we have an integer variable, say `x`, we can assign the address of `x` to a pointer variable of type `int *`. For example, if `p` is a pointer to an integer, we can write `p=&x`. After executing this instruction the value of `p` changes to the address of the variable `x`. Even though an address looks like an integer, it is illegal to write `p=2000`. Note that, if we declare `p` as a pointer to an *integer*, we can not store the address of a *float* variable in `p`.

The second major operation is the `*` operation. The expression `*p` refers to the location or variable to which `p` is pointing to (or in other words, `*p` refers to the location whose address is the current value of the pointer). For example, after executing `p=&x`, the expression `*p` has similar meaning as the expression `x`.

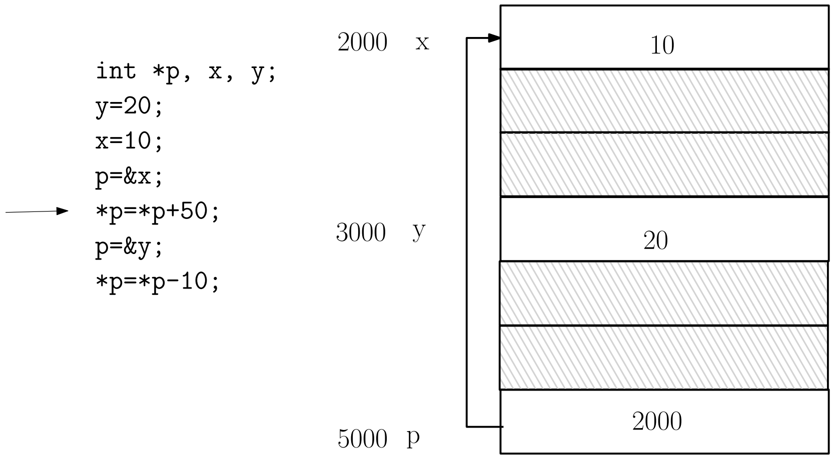
Consider the following example.

```
int *p, x, y;
y=20;
x=10;
p=&x;
*p=*p+50;
p=&y;
*p=*p-10;
```

When this program starts execution, the memory state diagram is as follows.

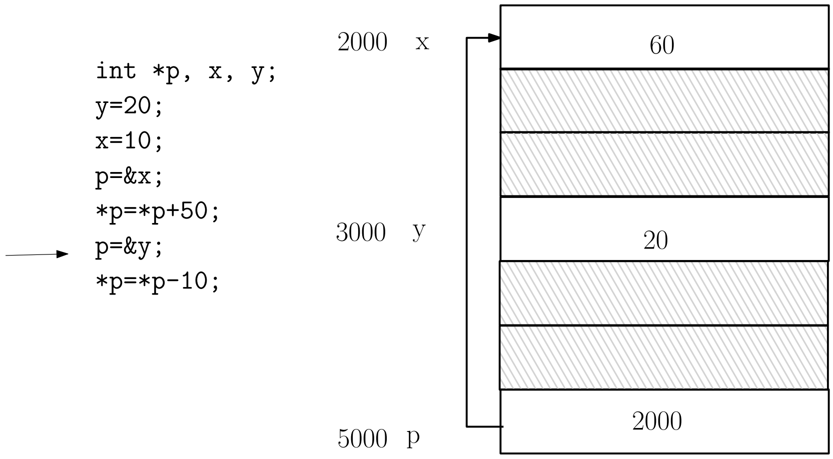


After executing `y=20`, the value of `y` changes to 20. After executing `x=10`, the value of `x` changes to 10. After executing `p=&x`, the address of the variable `x` is copied to the location of `p`. As per our previous figure, now, the value of `p` changes to 2000.



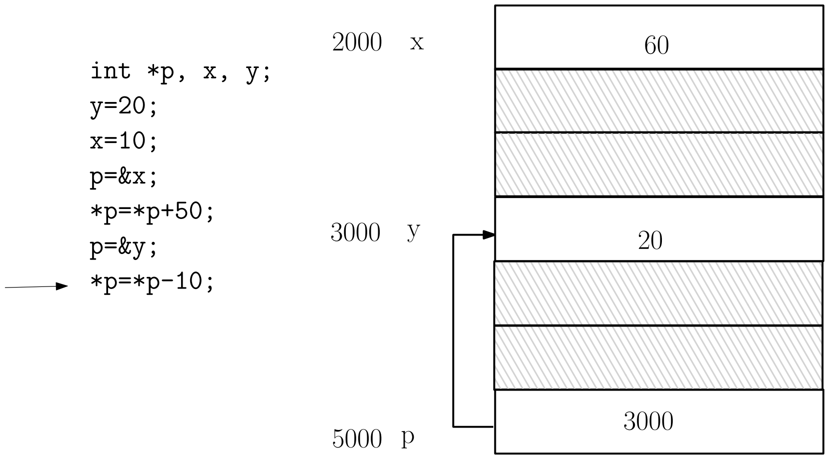
```
int *p, x, y;
y=20;
x=10;
p=&x;
*p=*p+50;
p=&y;
*p=*p-10;
```

Now, since 2000 is the value of `p`, the expression `*p` refers to the location with address 2000. Therefore, when the expression `*p+50` is evaluated in CPU, the result is the sum of the current value stored in location 2000 and the value 50, which is equal to 10+50=60. After executing the instruction `*p=*p+50`, the resultant value 60 is copied to location 2000. This is because, 2000 is the address stored as value of `p`. Note that, in effect the value of `x` changes to 60. At this stage, the memory state diagram is as given below.



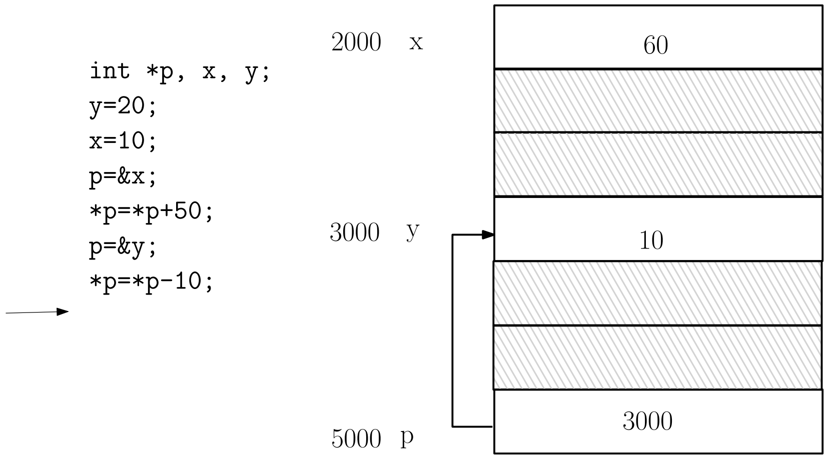
```
int *p, x, y;
y=20;
x=10;
p=&x;
*p=*p+50;
p=&y;
*p=*p-10;
```

After executing the instruction `p=&y`, the value of `p` is changed to the address of y, which is 3000. This is shown in the next figure.



```
int *p, x, y;
y=20;
x=10;
p=&x;
*p=*p+50;
p=&y;
*p=*p-10;
```

The next instruction is `*p=*p-10`. Now, since 3000 is the value of `p`, the expression `*p` refers to the location with address 3000. Therefore, when the expression `*p-10` is evaluated in CPU, the result is the difference of the current value stored in location 3000 and the value 10, which is equal to 20-10=10. After executing the instruction `*p=*p-10`, the resultant value 10 is copied to location 3000. This is because, 3000 is the address stored as value of `p`. Note that, in effect the value of `y` changes to 10. At this stage, the memory state diagram is as given below.

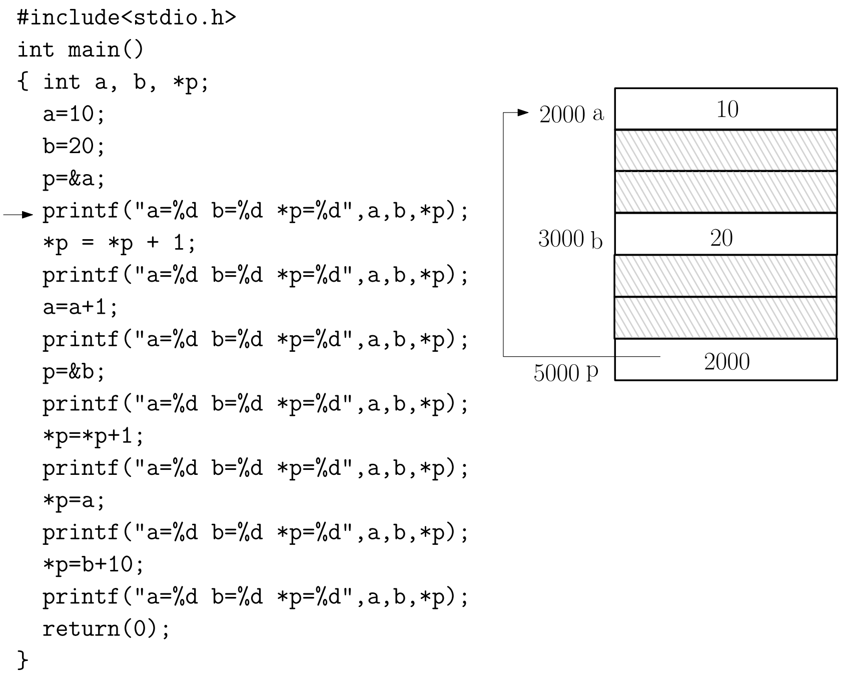


```
int *p, x, y;
y=20;
x=10;
p=&x;
*p=*p+50;
p=&y;
*p=*p-10;
```

Another example of usage of pointers is given below.

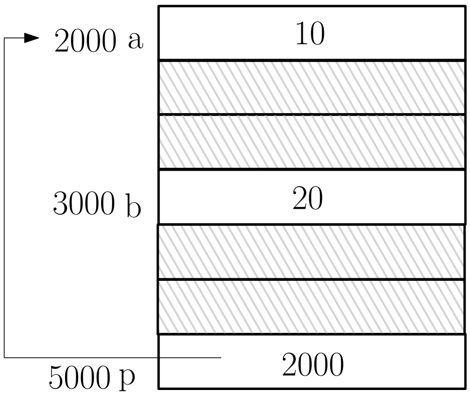
```
#include<stdio.h>
int main()
{ int a, b, *p;
  a=10;
  b=20;
  p=&a;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p = *p + 1;
  printf("a=%d b=%d *p=%d",a,b,*p);
  a=a+1;
  printf("a=%d b=%d *p=%d",a,b,*p);
  p=&b;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p=*p+1;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p=a;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p=b+10;
  printf("a=%d b=%d *p=%d",a,b,*p);
  return(0);
}
```

After executing the instructions up to p=&a;, the memory state diagram is as shown below.



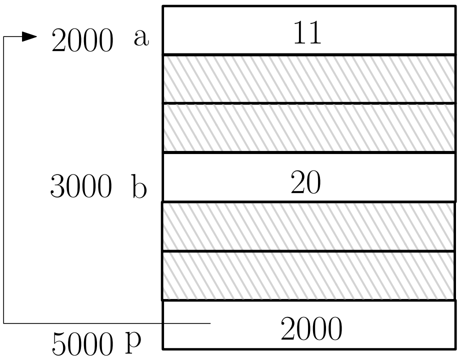
```
#include<stdio.h>
int main()
{ int a, b, *p;
  a=10;
  b=20;
  p=&a;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p = *p + 1;
  printf("a=%d b=%d *p=%d",a,b,*p);
  a=a+1;
  printf("a=%d b=%d *p=%d",a,b,*p);
  p=&b;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p=*p+1;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p=a;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p=b+10;
  printf("a=%d b=%d *p=%d",a,b,*p);
  return(0);
}
```

```
#include<stdio.h>
int main()
{
  int a, b, *p;
  a=10;
  b=20;
  p=&a;
  printf("a=%d b=%d *p=%d",a,b,*p);
→ *p = *p + 1;
  printf("a=%d b=%d *p=%d",a,b,*p);
  a=a+1;
  printf("a=%d b=%d *p=%d",a,b,*p);
  p=&b;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p=*p+1;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p=a;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p=b+10;
  printf("a=%d b=%d *p=%d",a,b,*p);

  return(0);
}
```

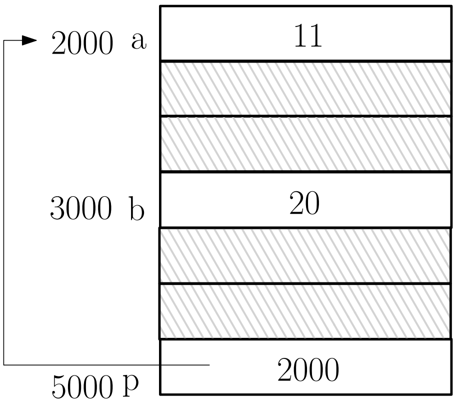


Now, output `a=10 b=20 *p=10` **is printed on screen.**

```
#include<stdio.h>
int main()
{
  int a, b, *p;
  a=10;
  b=20;
  p=&a;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p = *p + 1;
→ printf("a=%d b=%d *p=%d",a,b,*p);
  a=a+1;
  printf("a=%d b=%d *p=%d",a,b,*p);
  p=&b;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p=*p+1;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p=a;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p=b+10;
  printf("a=%d b=%d *p=%d",a,b,*p);

  return(0);
}
```

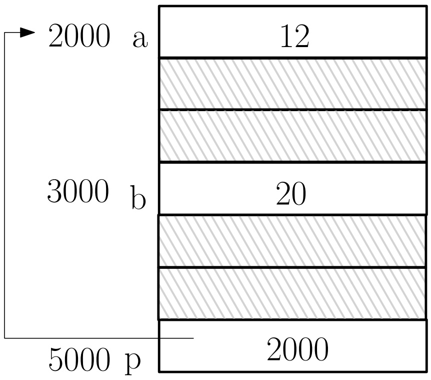


```
#include<stdio.h>
int main()
{
  int a, b, *p;
  a=10;
  b=20;
  p=&a;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p = *p + 1;
  printf("a=%d b=%d *p=%d",a,b,*p);
→ a=a+1;
  printf("a=%d b=%d *p=%d",a,b,*p);
  p=&b;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p=*p+1;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p=a;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p=b+10;
  printf("a=%d b=%d *p=%d",a,b,*p);

  return(0);
}
```

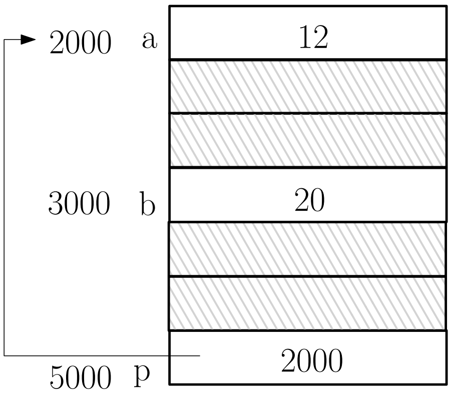


At this point `a=11 b=20 *p=11` **is printed on screen.**

```
#include<stdio.h>
int main()
{
  int a, b, *p;
  a=10;
  b=20;
  p=&a;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p = *p + 1;
  printf("a=%d b=%d *p=%d",a,b,*p);
  a=a+1;
→ printf("a=%d b=%d *p=%d",a,b,*p);
  p=&b;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p=*p+1;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p=a;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p=b+10;
  printf("a=%d b=%d *p=%d",a,b,*p);

  return(0);
}
```

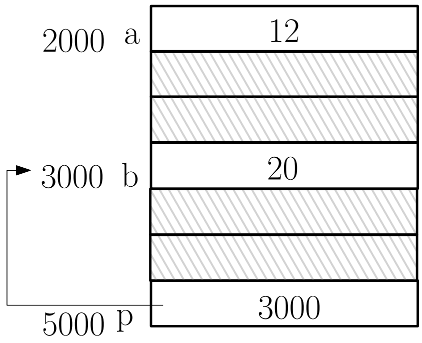


```
 #include<stdio.h>
 int main()
 {
   int a, b, *p;
   a=10;
   b=20;
   p=&a;
   printf("a=%d b=%d *p=%d",a,b,*p);
   *p = *p + 1;
   printf("a=%d b=%d *p=%d",a,b,*p);
   a=a+1;
   printf("a=%d b=%d *p=%d",a,b,*p);
→  p=&b;
   printf("a=%d b=%d *p=%d",a,b,*p);
   *p=*p+1;
   printf("a=%d b=%d *p=%d",a,b,*p);
   *p=a;
   printf("a=%d b=%d *p=%d",a,b,*p);
   *p=b+10;
   printf("a=%d b=%d *p=%d",a,b,*p);

   return(0);
 }
```

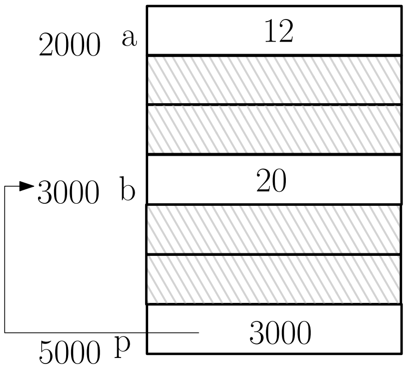


At this point `a=12 b=20 *p=12` is printed on screen.

```
#include<stdio.h>
int main()
{
  int a, b, *p;
  a=10;
  b=20;
  p=&a;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p = *p + 1;
  printf("a=%d b=%d *p=%d",a,b,*p);
  a=a+1;
  printf("a=%d b=%d *p=%d",a,b,*p);
  p=&b;
→ printf("a=%d b=%d *p=%d",a,b,*p);
  *p=*p+1;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p=a;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p=b+10;
  printf("a=%d b=%d *p=%d",a,b,*p);

  return(0);
}
```

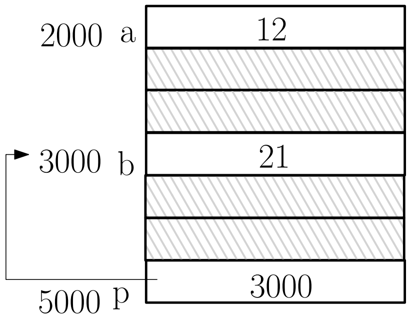


```
 #include<stdio.h>
 int main()
 {
   int a, b, *p;
   a=10;
   b=20;
   p=&a;
   printf("a=%d b=%d *p=%d",a,b,*p);
   *p = *p + 1;
   printf("a=%d b=%d *p=%d",a,b,*p);
   a=a+1;
   printf("a=%d b=%d *p=%d",a,b,*p);
   p=&b;
   printf("a=%d b=%d *p=%d",a,b,*p);
→  *p=*p+1;
   printf("a=%d b=%d *p=%d",a,b,*p);
   *p=a;
   printf("a=%d b=%d *p=%d",a,b,*p);
   *p=b+10;
   printf("a=%d b=%d *p=%d",a,b,*p);

   return(0);
 }
```

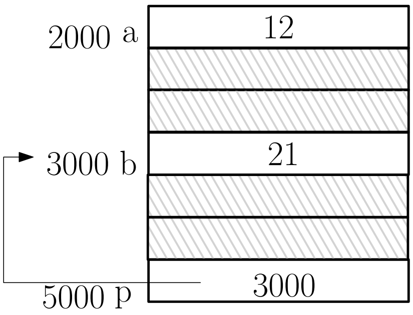


At this point `a=12 b=20 *p=20` **is printed on screen.**

```
#include<stdio.h>
int main()
{
  int a, b, *p;
  a=10;
  b=20;
  p=&a;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p = *p + 1;
  printf("a=%d b=%d *p=%d",a,b,*p);
  a=a+1;
  printf("a=%d b=%d *p=%d",a,b,*p);
  p=&b;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p=*p+1;
→ printf("a=%d b=%d *p=%d",a,b,*p);
  *p=a;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p=b+10;
  printf("a=%d b=%d *p=%d",a,b,*p);

  return(0);
}
```

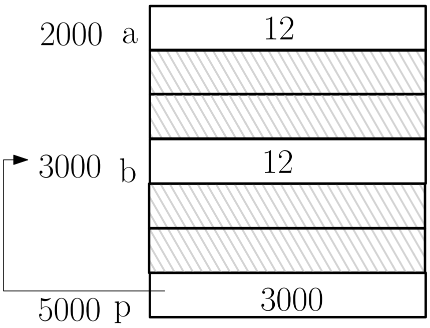


```
#include<stdio.h>
int main()
{
  int a, b, *p;
  a=10;
  b=20;
  p=&a;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p = *p + 1;
  printf("a=%d b=%d *p=%d",a,b,*p);
  a=a+1;
  printf("a=%d b=%d *p=%d",a,b,*p);
  p=&b;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p=*p+1;
  printf("a=%d b=%d *p=%d",a,b,*p);
→ *p=a;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p=b+10;
  printf("a=%d b=%d *p=%d",a,b,*p);

  return(0);
}
```

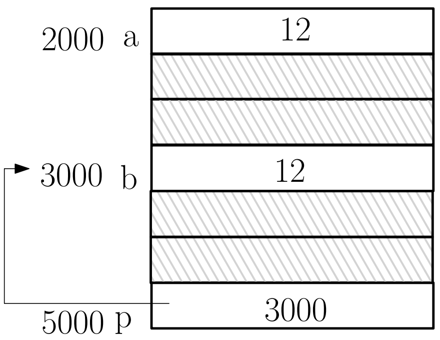


At this point `a=12 b=21 *p=21` is printed on screen.

```
#include<stdio.h>
int main()
{
  int a, b, *p;
  a=10;
  b=20;
  p=&a;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p = *p + 1;
  printf("a=%d b=%d *p=%d",a,b,*p);
  a=a+1;
  printf("a=%d b=%d *p=%d",a,b,*p);
  p=&b;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p=*p+1;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p=a;
→ printf("a=%d b=%d *p=%d",a,b,*p);
  *p=b+10;
  printf("a=%d b=%d *p=%d",a,b,*p);

  return(0);
}
```

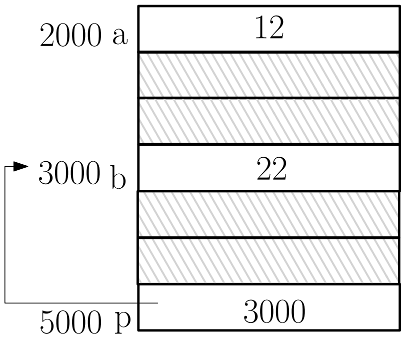


```
#include<stdio.h>
int main()
{
  int a, b, *p;
  a=10;
  b=20;
  p=&a;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p = *p + 1;
  printf("a=%d b=%d *p=%d",a,b,*p);
  a=a+1;
  printf("a=%d b=%d *p=%d",a,b,*p);
  p=&b;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p=*p+1;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p=a;
  printf("a=%d b=%d *p=%d",a,b,*p);
→ *p=b+10;
  printf("a=%d b=%d *p=%d",a,b,*p);

  return(0);
}
```

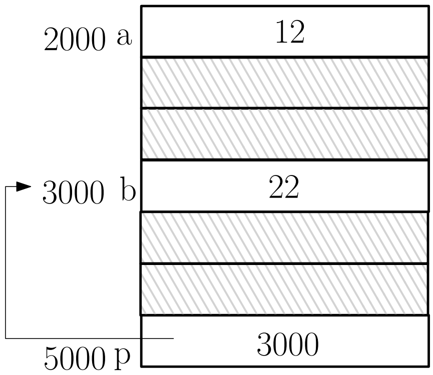


At this point a=12 b=12 *p=12 **is printed on screen.**

```
#include<stdio.h>
int main()
{
  int a, b, *p;
  a=10;
  b=20;
  p=&a;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p = *p + 1;
  printf("a=%d b=%d *p=%d",a,b,*p);
  a=a+1;
  printf("a=%d b=%d *p=%d",a,b,*p);
  p=&b;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p=*p+1;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p=a;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p=b+10;
→ printf("a=%d b=%d *p=%d",a,b,*p);

  return(0);
}
```



```
#include<stdio.h>
int main()
{
  int a, b, *p;
  a=10;
  b=20;
  p=&a;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p = *p + 1;
  printf("a=%d b=%d *p=%d",a,b,*p);
  a=a+1;
  printf("a=%d b=%d *p=%d",a,b,*p);
  p=&b;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p=*p+1;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p=a;
  printf("a=%d b=%d *p=%d",a,b,*p);
  *p=b+10;
  printf("a=%d b=%d *p=%d",a,b,*p);

→ return(0);
}
```



At this point `a=12 b=22 *p=22` **is printed on screen.**