

Tutorial 8

Pointers to pointers

Jiawei Li (Michael)

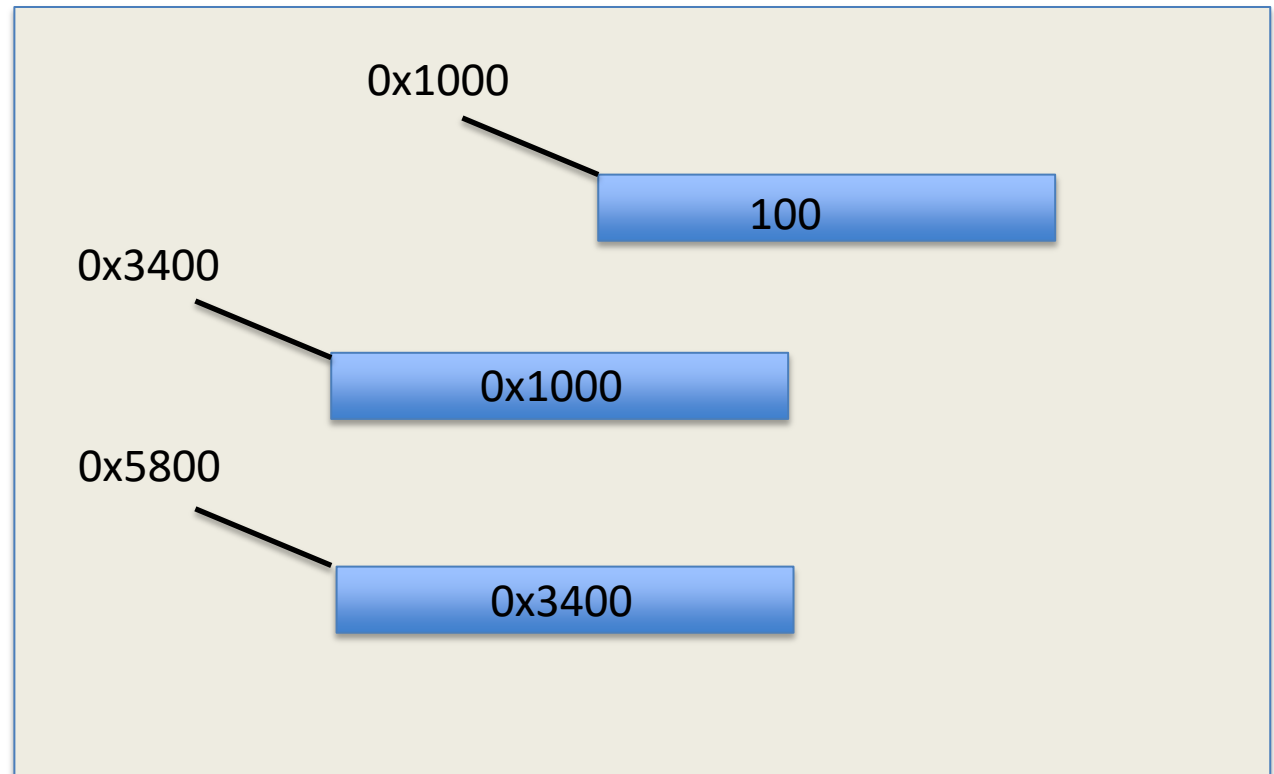
Office hours: Tuesday 3:00-5:30pm

Office: PMB426

Email: jiawei.li@nottingham.edu.cn

Define a variable

- `int a;`
- `int *p;`
- `p = &a;`
- `*p = 100;`
- `int **ptr;`
- `ptr=&p;`
- `**ptr=101;`



Declaration vs. dereference

```
int* p1;  
int *p2;  
int * p3;
```

```
int* Func1(int a, int* arr);  
int *Func2(int a, int* arr);
```

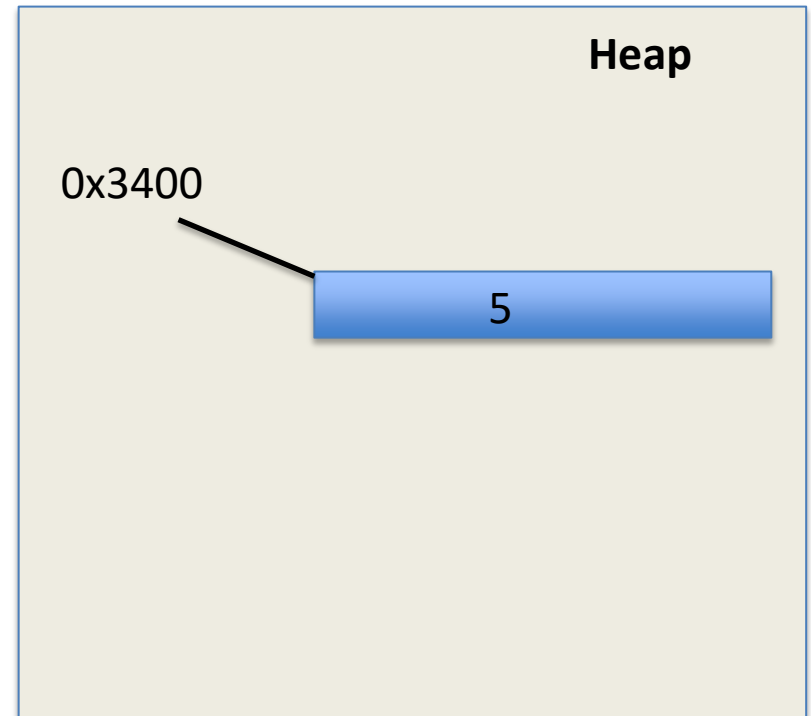
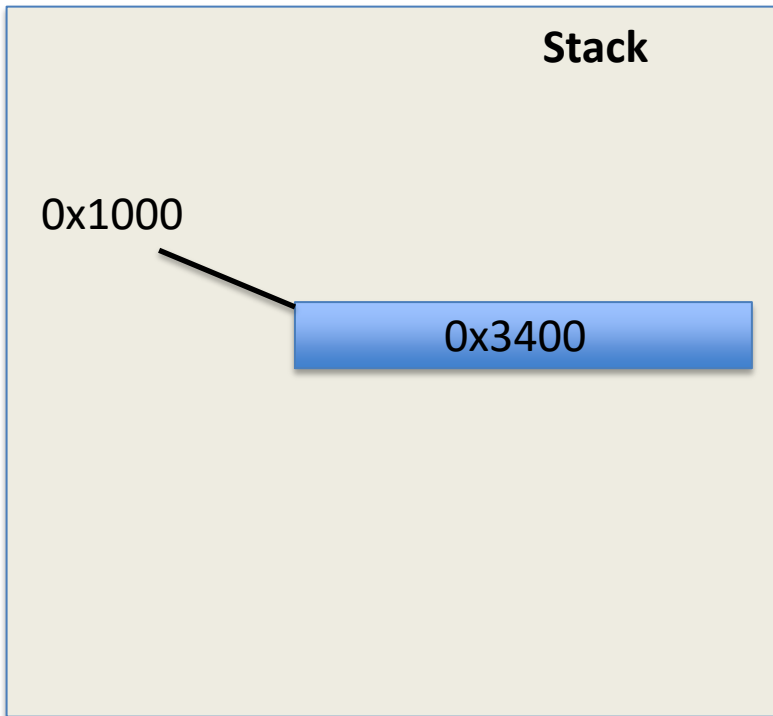
```
int *p;  
p = &value;  
*p = 5;
```

```
int *p = &value;  
*p = 5;
```

```
int *p = malloc(sizeof(int));  
*p = 5;
```

malloc() and free()

```
int *p;  
p = malloc(sizeof(int));  
*p = 5;  
free(p);
```



Example 1

```
#include<stdio.h>
int main(int argc, char *argv[])
{
    int a = 1;
    int b = 2;
    int *p1 = &a;
    int *p2 = &b;
    int **pp1 = &p1;
    printf("%d      %p      %p\n", **pp1, *pp1, pp1);

    *pp1 = p2; // pp1 = &p2;
    printf("%d      %p      %p\n", **pp1, *pp1, pp1);
    printf("%p      %p\n", &a, &b);
    printf("%p      %p\n", p1, p2);

    return 0;
}
```

Why pointers to pointers?

When we use "pass by pointer" to pass a pointer to a function, only a copy of the pointer is passed to the function.

In most cases, this does not present a problem. But problem comes when you modify the pointer inside the function. Instead of modifying the variable, you are only modifying a copy of the pointer and the original pointer remains unmodified, that is, it still points to the old variable.

```
int g_n = 42;

void example_ptr()
{
    int n = 23;
    int* pn = &n;
    printf("%d", *pn);
    func_ptr(pn);
    printf("%d", *pn);
}

void func_ptr(int* pp)
{
    pp = &g_n;
}
```

How to revise this program?

```
#include<stdio.h>
void swap(int* arg1, int* arg2)
{
    int* value = NULL;
    value = arg1;
    arg1 = arg2;
    arg2 = value;
}

int main()
{
    int a = 1;
    int b = 2;
    int *p1 = &a;
    int *p2 = &b;
    swap(p1, p2);
    printf("%d, %d\n", *p1, *p2);
}
```

Why pointers to pointers?

When we need to modify a pointer inside a function, we should pass the address of the pointer to the function.

```
void append(IntList **list, int value);
```

```
IntList *start;  
append(&start, 5);
```

```
int g_n = 42;  
  
void example_ptr()  
{  
    int n = 23;  
    int *pn = &n;  
    printf("%d", *pn);  
    func_ptr_to_ptr(&pn);  
    printf("%d", *pn);  
}  
  
void func_ptr_to_ptr(int **pp)  
{  
    *pp = &g_n;  
}
```


Frequent mistakes

- Not assigning memory.

```
int **ptr1;  
*ptr1 = malloc(sizeof(int));
```

```
int *ptr2= malloc(sizeof(int));  
int **ptr1 = &ptr2;
```

Frequent mistakes

- Misuse pointers in function calls

```
void func( int** ptr)
{
    *ptr = x; // assume that x is a pointer;
              // Does this code work?
    // ptr = &x;
              // what is the difference?
}

int main()
{
    int *ptr = NULL;
    func(&ptr);

    // int **ptr = NULL;
    // func(ptr);
    ...
}
```

Example 2

```
void func1(int *x, int *y);
void func2(int **x, int *y);
int main(void)
{
    int a;
    int *p1 = NULL;

    func1(p1, &a);
    printf("%d  %p\n", a, p1);

    func2(&p1, &a);
    printf("%d  %p\n", a, p1);

    return 0;
}
```

```
void func1(int *x, int *y)
{
    *y = 1;
    x = y;
}

void func2(int **x, int *y)
{
    *y = 2;
    *x = y;
}
```

Task: Complete a Trie

You are given a partially-completed program which implements a Trie. ([trie_incomplete.c](#) in the Tutorial section)

Your task is to complete the function [trie_contains\(\)](#) so that the program print **"Test passed"** when run.

The [trie_contains](#) function should check if a given string is present in the trie or not.

[assert\(expression\)](#) is a macro in C. If [expression](#) is true, it does nothing. If [expression](#) is false, it terminates the program.