Computer Programming & Problem Solving

CS100

Mrs Sanga G. Chaki

Department of Computer Science and Engineering

National Institute of Technology, Goa

April, 2023

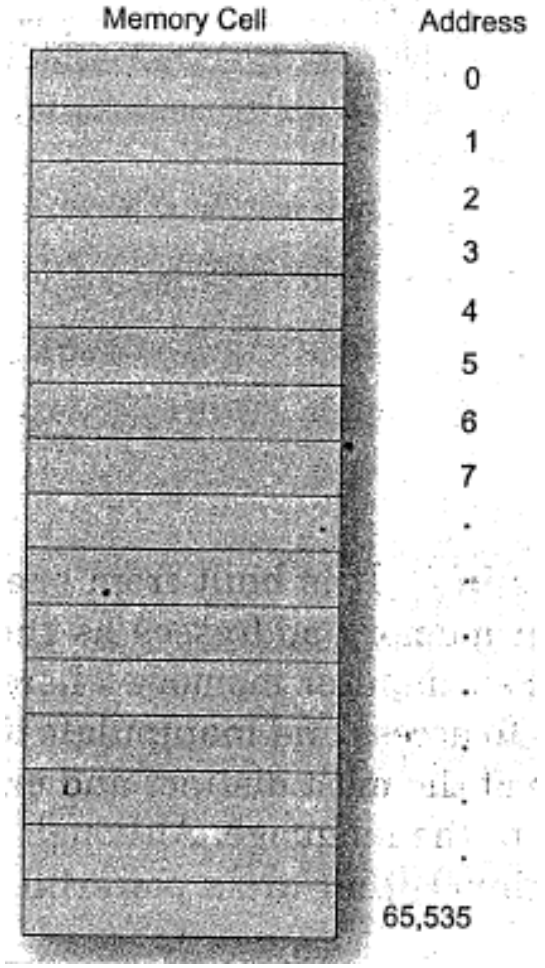
Pointer Basics

New Terms

1. Memory allocation
2. Pointers
3. Pointer notations
    1. Address of (&)
    2. Value at address (*)
4. Pointer to a pointer

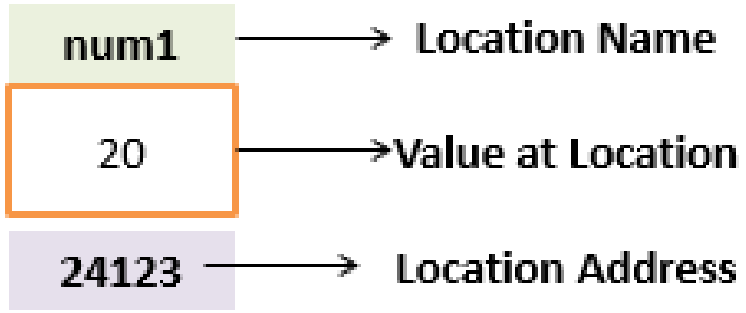
Memory – Bits, bytes, how variables are stored

1. Computer's memory is a sequential collection of storage cells
2. Each cell is called a byte
3. A byte has an associated address
4. Addresses are numbered consecutively – start from 0
5. Last address depends on memory size



Memory – Bits, bytes, how variables are stored

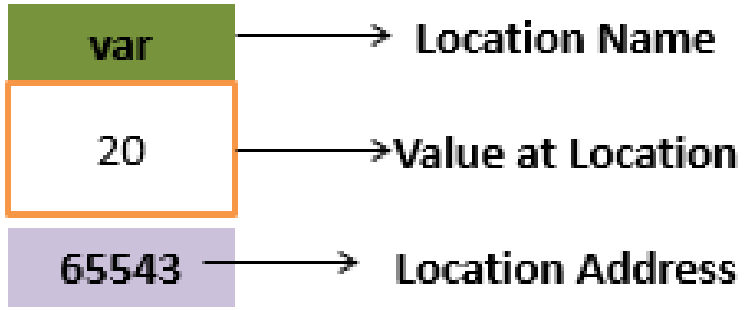
1. Whenever we declare a variable, the system allocates some memory to hold that variable – this means, a memory location is allocated with unique address number.



2. So, when we say: **int a = 10;** this means:

a) System find an appropriate location for a

b) Stores value of a in that location

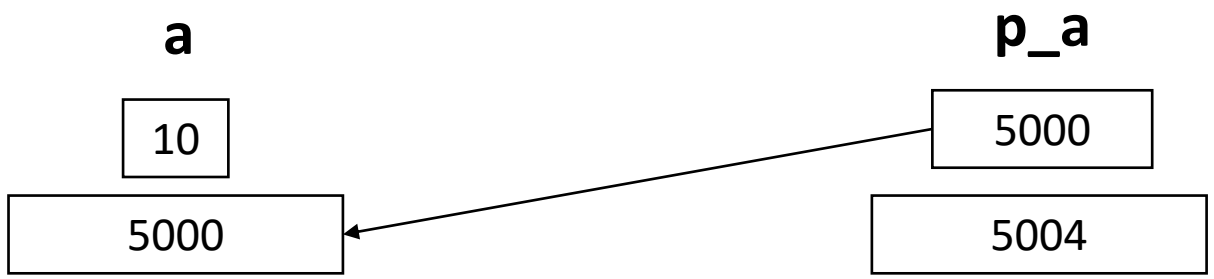


3. A variable can occupy multiple bytes – its address is the address of the first byte.

4. Location name = variable name

Memory – Bits, bytes, how variables are stored

1. System associates name **a** with address **5000**
2. We can access the value 10, by either using the name a, or the address 5000
3. Since memory addresses are numbers, they can be assigned to other variables and stored in memory.
4. Such a variable that contains address of another variable is called a **pointer variable**



Pointers - Notations

- **&** → **Address of** operator
- * → **Value at address** operator

```
main( )
{
        int  i = 3 ;

        printf ( "\nAddress of i = %u", &i ) ;
        printf ( "\nValue of i = %d", i ) ;
        printf ( "\nValue of i = %d", *( &i ) ) ;
}
```

How to create pointers to variables?

```
main( )
{
      int  i = 3 ;

      printf ( "\nAddress of i = %u", &i ) ;
      printf ( "\nValue of i = %d", i ) ;
      printf ( "\nValue of i = %d", *( &i ) ) ;
}
```

How to collect the address of variable "i" in another variable?

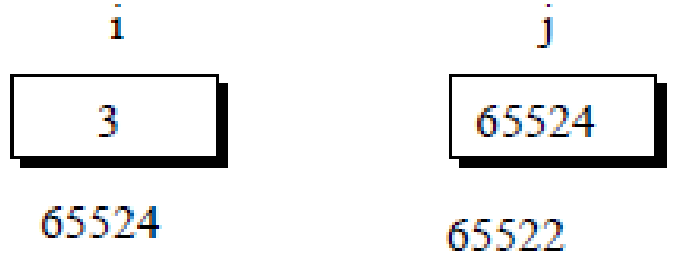
```
j = &i ;
```

What is this "j"? It contains the address of another variable. – So, it is a pointer.
How to declare it?

```
int  *j ;
```

This declaration tells the compiler that j will be used to store the address of an integer value.

j points to an integer.



Pointers – Basic Concept

1. A pointer is a variable that stores the memory address of another variable as its value.
2. A pointer variable points to a data type of the same type, and is created with the * operator.
3. The address of the variable you are working with is assigned to the pointer

Pointers – Example code

```
main( )
{
        int  i = 3 ;
        int  *j ;

        j = &i ;
        printf ( "\nAddress of i = %u", &i ) ;
        printf ( "\nAddress of i = %u", j ) ;
        printf ( "\nAddress of j = %u", &j ) ;
        printf ( "\nValue of j = %u", j ) ;
        printf ( "\nValue of i = %d", i ) ;
        printf ( "\nValue of i = %d", *( &i ) ) ;
        printf ( "\nValue of i = %d", *j ) ;
}
```

```
Address of i = 65524
Address of i = 65524
Address of j = 65522
Value of j = 65524
Value of i = 3
Value of i = 3
Value of i = 3
```

Pointer to pointer – Example code

```
main( )
{
      int  i = 3, *j, **k ;

      j = &i ;
      k = &j ;
      printf ( "\nAddress of i = %u", &i ) ;
      printf ( "\nAddress of i = %u", j ) ;
      printf ( "\nAddress of i = %u", *k ) ;
      printf ( "\nAddress of j = %u", &j ) ;
      printf ( "\nAddress of j = %u", k ) ;
      printf ( "\nAddress of k = %u", &k ) ;
      printf ( "\nValue of j  = %u", j ) ;
      printf ( "\nValue of k  = %u", k ) ;
      printf ( "\nValue of i  = %d", i ) ;
      printf ( "\nValue of i  = %d", * ( &i ) ) ;
      printf ( "\nValue of i  = %d", *j ) ;
      printf ( "\nValue of i  = %d", **k ) ;
}
```

```
Address of i = 65524

Address of i = 65524
Address of i = 65524
Address of j = 65522
Address of j = 65522
Address of k = 65520
Value of j  = 65524
Value of k  = 65522
```

```
Value of i  = 3
Value of i  = 3
Value of i  = 3
Value of i  = 3
```

Call by Value

Call by Value

1. Till now, whatever function examples we have seen use call by value.
2. So, what happens in call by value?
3. The ‘value’ of each of the actual arguments in the calling function is copied into corresponding formal arguments of the called function.
4. The changes made to the formal arguments in the called function have no effect on the values of actual arguments in the calling function

Call by Value - Example

```c
#include <stdio.h>
void func(int a, int b){
    int t;
    t=a;
    a=b;
    b=t;
    printf("\nIn func: a = %d, b = %d", a, b);
}

int main() {
    int a = 10, b=20;
    printf("\nIn main: a = %d, b = %d", a, b);
    func(a,b);
    printf("\nIn main: a = %d, b = %d", a, b);

    return 0;
}
```

Output

```
/tmp/lEog0alxua.o
In main: a = 10, b = 20
In func: a = 20, b = 10
In main: a = 10, b = 20
```

Call by Reference

Call by Reference

1. In call by reference instead of passing the value of a variable, we pass the location number (also called address) of the variable to a function.
2. We use pointers.
3. The addresses of actual arguments in the calling function are copied into formal arguments of the called function.
4. This means that using these addresses we would have an access to the actual arguments and hence we would be able to manipulate them

Call by Reference – Example 1

```c
#include <stdio.h>
int add_v(int i){
    int r = i+10;
    return r;
}
int add_r1(int *j){
    int s = *j+10;
    return s;
}
void add_r2(int *k){
    *k = *k+10;
}
int main() {
    int x = 10;
    int a,b;
    a = add_v(x);
    printf("a = %d\n",a);
    b = add_r1(&x);
    printf("b = %d\n",b);
    printf("x = %d\n",x);
    add_r2(&x);
    printf("x = %d",x);
    return 0;}
```

```
/tmp/1pvFSgu8uC.o
a = 20
b = 20
x = 10
x = 20
```

Call by Reference – Example 2

```c
#include <stdio.h>
void func(int *i, int *j){
    printf("\nIn func: a = %d, b = %d", *i, *j);
    int t;
    t = *i;
    *i = *j;
    *j = t;
    printf("\nIn func: a = %d, b = %d", *i, *j);
}
int main() {
    int a = 10, b=20;
    printf("\nIn main: a = %d, b = %d", a, b);
    func(&a,&b);
    printf("\nIn main: a = %d, b = %d", a, b);

    return 0;
}
```

Output

```
/tmp/V9HoLjcqgS.o
In main: a = 10, b = 20
In func: a = 10, b = 20
In func: a = 20, b = 10
In main: a = 20, b = 10
```

Return more than 1 value from function - Call by Reference

```
main( )
{
        int  radius ;
        float  area, perimeter ;

        printf ( "\nEnter radius of a circle " ) ;
        scanf ( "%d", &radius ) ;
        areaperi ( radius, &area, &perimeter ) ;

        printf ( "Area = %f", area ) ;
        printf ( "\nPerimeter = %f", perimeter ) ;
}

areaperi ( int  r, float  *a, float  *p )
{
        *a = 3.14 * r * r ;
        *p = 2 * 3.14 * r ;
}
```

What is the output?

What is happening here?

Mixed call –

1. Pass the value of radius
2. Pass the addresses of area and perimeter
3. **overcome the limitation of the return statement**