CMP 1001

Introduction to programming

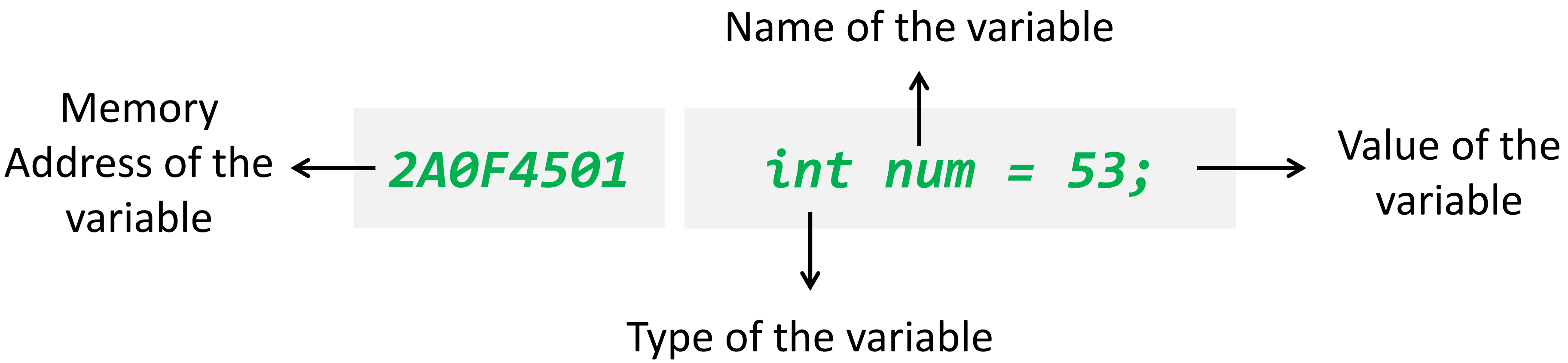
Part 8: Pointers, Passing values to functions

by

Assist. Prof. M. Şükrü Kuran

ADDRESS OF VARIABLES

Where do the variables located in the Memory?



Name of the variable

Memory Address of the variable

2A0F4501

int num = 53;

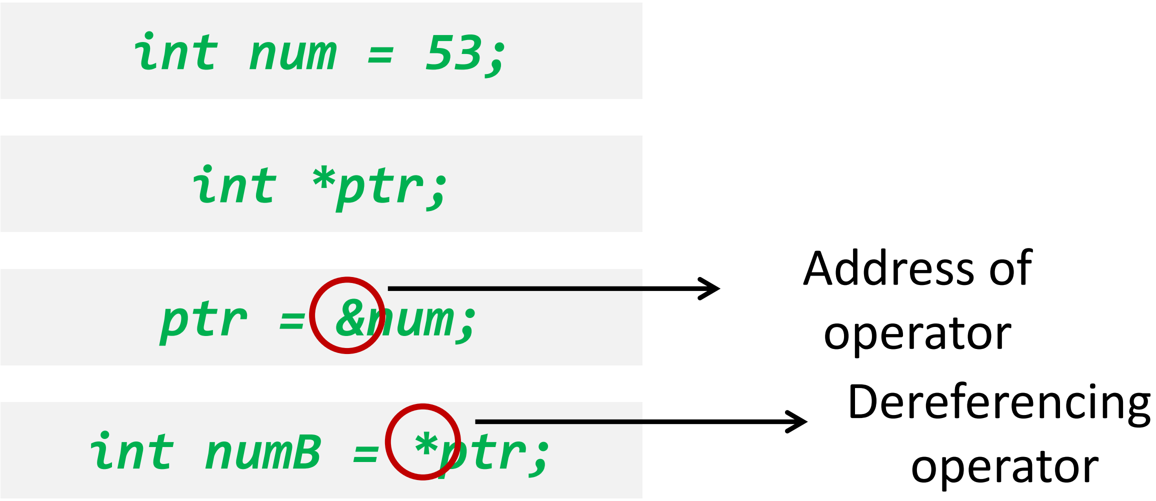
Value of the variable

Type of the variable

```
int num = 53;
std::cout << “Address of num is ” << &num;
```

POINTER

Variable for keeping the address of a variable



```
int num = 53;
int *ptr;
ptr = &num;
int numB = *ptr;
```

Address of operator

Dereferencing operator

POINTER

Pointer Types

Variable Type	Size (in bits)
int *	32 or 64
short *	32 or 64
byte *	32 or 64
long *	32 or 64
float *	32 or 64
double *	32 or 64
long double *	32 or 64

NOTE: For every type T, there exists a type “pointer to T” (including structs).

POINTER

Example

Example:

Write a program that reads two double numbers (numA, numB). Define pointers to both of these double numbers. Then, calculate the following via the pointers to these numbers and print out the result to the screen.

- **Addition**
- **Subtraction**
- **Multiplication**
- **Division**

PASSING VALUES TO FUNCTIONS

Call-by-value

When you pass a variable to a function, C++ sends a/the **COPY/VALUE** of the variable.

Call-by-reference

When you pass a pointer of a variable to a function, C++ sends the **ADDRESS/REFERENCE** of the variable.

Example:
Study the example for the difference between Call-by-value and Call-by-reference implementations of the swap function.

PASSING VALUES TO FUNCTIONS

Example:

Write a program that takes three integer numbers, numA, numB, and numC from the user.

Then, write a function called ‘order’ that takes three integer numbers and three integer pointers and returns nothing as seen below.

```
void order(int numA, int numB, int numC, int *minp, int *maxp, int *medp)
```

This function will calculate the minimum, maximum, and median of these three numbers and return these three results via the minp, maxp, medp pointers to the main function.

Finally, print these maximum, minimum, and median values to the screen.

OPERATIONS WITH POINTERS

Arithmetical operations work with pointer variables.

```
int x = 15;
int *p = &x;
*(p+1) = 12;
*(p-2) = 13;
*(p/2) = 5;
```

Address	Name	Value
1200BA00	x	15
1200BA01		12
1200B9FE		13
...		5

POINTERS & ARRAYS

Relationship between Pointers & Arrays

Well, actually arrays are pointers in C++

```
int a[5];
int *p = &a;
*(a+1) = 91;
*(a+2) = 92;
*(a+3) = 93;
```

Address	Name	Alt.Name	Value
13000100	*(a+0)	a[0]	
13000101	*(a+1)	a[1]	91
13000102	*(a+2)	a[2]	92
13000103	*(a+3)	a[3]	93
13000104	*(a+4)	a[4]	

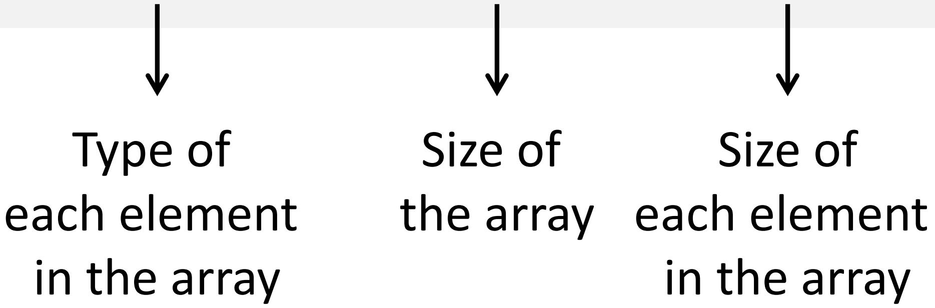
DYNAMIC MEMORY ALLOCATION (MALLOC)

```
#include <cstdlib>
```

Allocating memory spaces via pointers.

```
int array[10];
```

```
int *array = (int*)malloc(10*sizeof(int));
```



Type of each element in the array

Size of the array

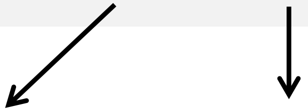
Size of each element in the array

Address	Name	Value
Address 0	*(array+0)	
Address 1	*(array+1)	
...		
Address 8	*(array+8)	
Address 9	*(array+9)	

DYNAMIC MEMORY ALLOCATION (REALLOC)

```
int *array = (int*)malloc(10*sizeof(int));
```

```
array = (int*)realloc(array, 15*sizeof(int));
```



Pointer to the memory block previously allocated

New size of the array

DYNAMIC MEMORY ALLOCATION (CALLOC)

Memory allocation and clearing the memory.

```
int array[10];
```

```
int *array = (int*)calloc(10, sizeof(int));
```

Type of each element in the array

Size of the array

Size of each element in the array

Address	Name	Value
`Address 0`	*(array+0)	**0**
`Address 1`	*(array+1)	**0**
`...`		
`Address 8`	*(array+8)	**0**
`Address 9`	*(array+9)	**0**

DYNAMIC MEMORY ALLOCATION

Function	Description
malloc	allocates the specified number of bytes
realloc	increases or decreases the size of the specified block of memory. Reallocates it if needed
calloc	allocates the specified number of bytes and initializes them to zero
free	releases the specified block of memory back to the system

```
free(array);
```

NOTE: If you declare a dynamic array with malloc, calloc, realloc; at the end of the program you MUST free this memory via free

DYNAMIC MEMORY ALLOCATION

Example:

Write a program that declares an integer array of size 8 using dynamic memory allocation. Then, the program keeps asking for numbers from the user to put inside the array until a negative number is entered.

If the user enters more than 8 numbers, the program MUST increase the size of the array ONE BY ONE using realloc method.

Finally, the program will write all the numbers inside the array.

CMP-1001

POINTER TO POINTERS

Address of a Pointer Variable

2A0F4501	`int num = 53;`
3CCC0002	`int *ptr = &num;`
10000602	`int **pptr = &ptr;`

POINTER TO POINTERS IN 2D ARRAYS

```
int array2D[10][5];
```

```
int **array2D = (int**)malloc(10*sizeof(int*));
```

```
array2D[0] = (int*)malloc(5*sizeof(int));
```

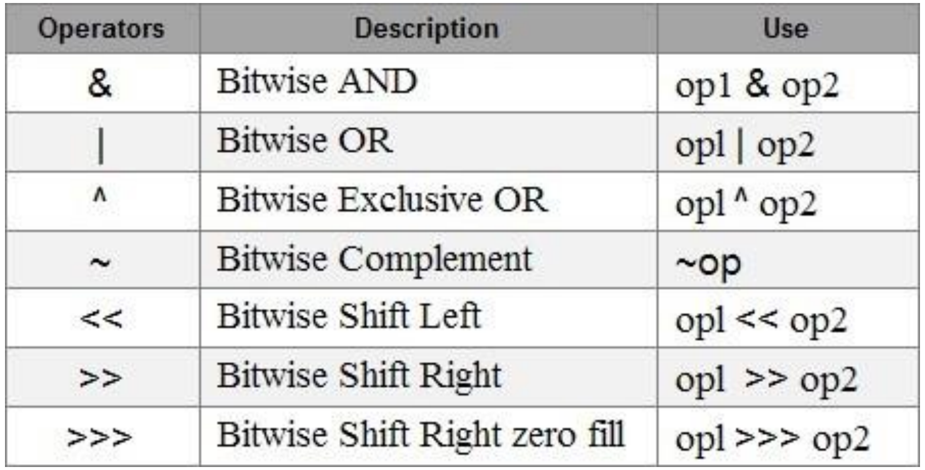
```
array2D[1] = (int*)malloc(5*sizeof(int));
```

```
...
```

```
array2D[4] = (int*)malloc(5*sizeof(int));
```

COMING SOON…

Next week on CMP 1001



Operators	Description	Use
&	Bitwise AND	op1 & op2
|	Bitwise OR	opl | op2
^	Bitwise Exclusive OR	opl ^ op2
~	Bitwise Complement	~op
<<	Bitwise Shift Left	opl << op2
>>	Bitwise Shift Right	opl >> op2
>>>	Bitwise Shift Right zero fill	opl >>> op2

Bitwise operators, files