C++ Programming
Lesson 2

TINPRO02-5B

HOGESCHOOL ROTTERDAM

PROGRAMMA

- Recap
- Pointers
  - Address operator
  - References
  - ~~Overloading~~

Recap Lesson 1

- Arrays
- For, while loops, if then else, switch
- Struct
- Enum

It’s testing time!

Wat hebben we vorige keer gedaan?

Ga naar: www.socrative.com ->login Student-> Room Name:

PIVAO123

Pointers and References

- In C++ objects reside at a specific address in memory
- Objects can be accessed if you know its address and its type

Computer		Programmers		
Address	Content	Name	Type	Value
90000000	00			
90000001	00	sum	int (4 bytes)	000000FF(255_{10})
90000002	00			
90000003	FF			
90000004	FF	age	short (2 bytes)	FFFF(-1_{10})
90000005	FF			
90000006	1F	averge	double (8 bytes)	1FFFFFFFFFFFFFFF ($4.45015E-308_{10}$)
90000007	FF			
90000008	FF			
90000009	FF			
9000000A	FF			
9000000B	FF			
9000000C	FF			
9000000D	FF			
9000000E	90	ptrSum	int* (4 bytes)	90000000
9000000F	00			
90000010	00			
90000011	00			

Note: All numbers in hexadecimal

Can I access external variables from a function?

- What happens when a function is called with some variables as parameters?
- Variables are normally passed “by value”. A copy of the variable is created for the function and dies with it.

```
void function (int) ;
void main()
{int a=3 ;
 function(a) ;
}
void function(int x)
{ x=x+1 ; }
```

Can I access external variables from a function?

- We could use GLOBAL variables that can be altered from any function… but this forces us to declare everything as global!
- or we could tell the compiler that the variables must be passed by “by reference” instead of a copy

```
void function (int &) ;
void main()
{int a=3 ;
 function(a) ;
}
void function(int &x)
{ x=x+1 ; }
```

What si a reference? How is it implemented?

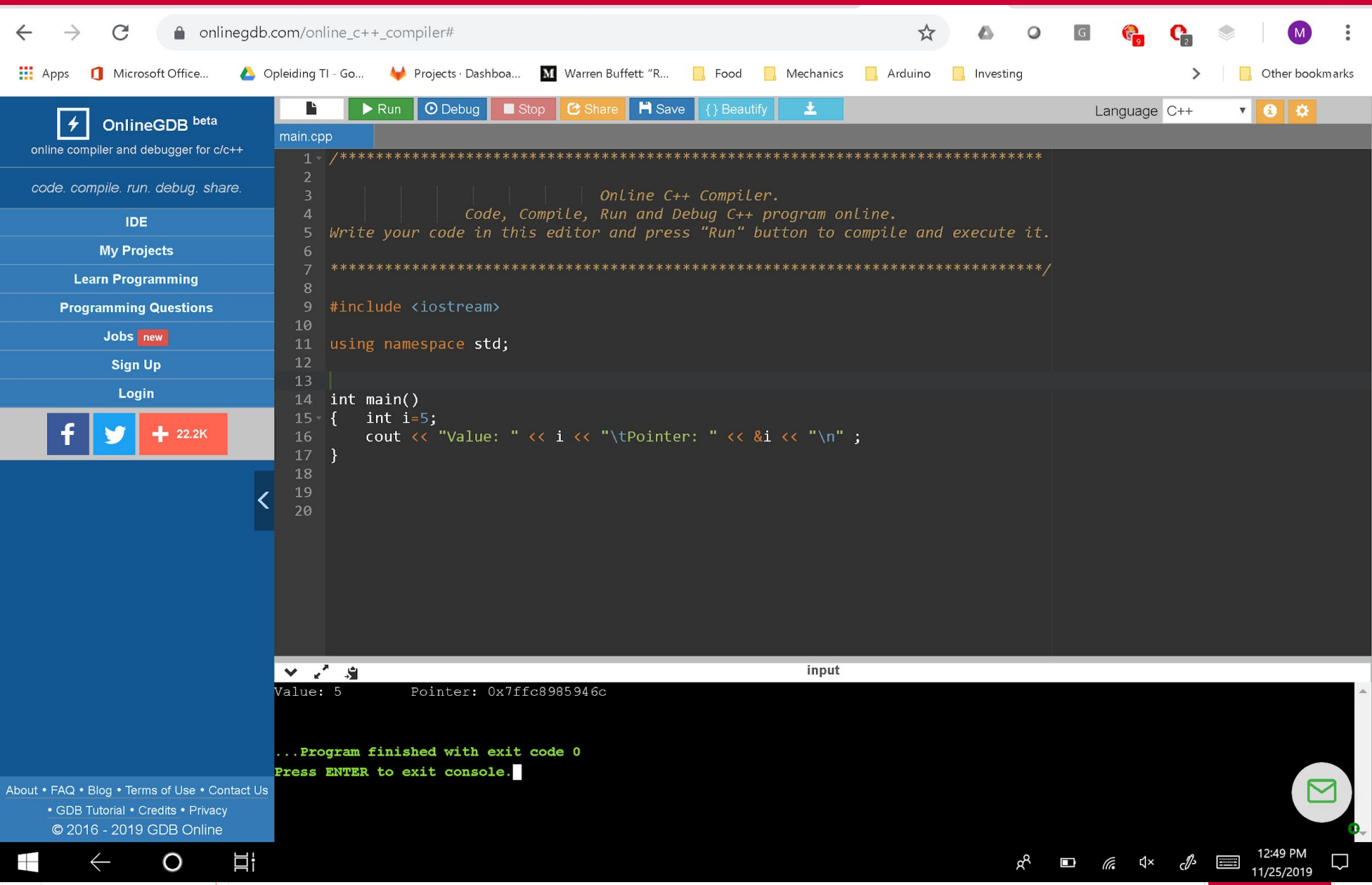
- A reference (int &x=a) is a new name for the same variable (area of memory).
- “x” is an alias for the variable “a” and they share the same memory area.
- This gives us a mechanism to address a variable without referring to its name. It also allows to “pierce through” the variable own scope [as the scope of “a” would naturally be the main(), while we are modifying its values outside its own scope]
- This is both dangerous and powerful but it hints at something deeper…

Pointers *

- A cleaner way, would be to to explicitly pass the memory address where the variable is residing to the function.
- The “address operator” **&** gives us the address of a variable, its **pointer**

```
// Declare a variable and prints out both its value and
its memory address (pointer)
int main(){
int i=5;
cout << "Value: " << i << "\tPointer: " << &i << "\n" ;
}
```

The result...


onlinegdb.com/online_c++_compiler#
Apps
Microsoft Office...
Opleiding TI - Go...
Projects · Dashboa...
Warren Buffett: "R...
Food
Mechanics
Arduino
Investing
Other bookmarks
OnlineGDB beta
online compiler and debugger for c/c++
code. compile. run. debug. share.
IDE
My Projects
Learn Programming
Programming Questions
Jobs new
Sign Up
Login
22.2K
Run
Debug
Stop
Share
Save
{} Beautify
Language C++
main.cpp
/***
Online C++ Compiler.
Code, Compile, Run and Debug C++ program online.
Write your code in this editor and press "Run" button to compile and execute it.
***/
#include <iostream>
using namespace std;
int main()
{ int i=5;
cout << "Value: " << i << "\tPointer: " << &i << "\n" ;
}
input
Value: 5 Pointer: 0x7ffc8985946c
...Program finished with exit code 0
Press ENTER to exit console.
About • FAQ • Blog • Terms of Use • Contact Us
• GDB Tutorial • Credits • Privacy
© 2016 - 2019 GDB Online
12:49 PM
11/25/2019

Pointers *

- Where can we, actually, store these pointers?
- We can define a “pointer variable” which can store a **memory address** occupied by a variable.

```
// Declare a pointer variable called iPtr that can be used to point at an int (an int pointer)
int *iPtr; // a pointer to int (it is currently meaningless as it has not been initialized!)
double *dPtr;  // Declare a pointer to a double (not-initialized)
```

Pointers

- How do we set the actual address of the pointer?
- We need a variable and then we use the *address operator* **&**!

```
// Declare a pointer to int variables, declares an int and
sets the pointer
int *p1;  // pointer p1 is not initialized!!
int i=5;    // int i is created and set to 5. Its memory (4
bytes)is allocated and set to the correct value
p1=&i ; // the address operator & applied to the variable i
returns the variable adress that is used to set the pointer.
```

De adres operator: &

- The address operator returns the address in the memory of the given variable
- It is just an operator, like +, -, *, etc

```
#include <iostream>
using namespace std;
int main() {
    int number = 5;
    cout<<"The value of number is "<< number<<endl;
    cout<<"The  address is "<< &number<<endl;
    return 0;
}
```

Address operator &

- When you declare a pointer variable, its content is not initialized->it contains an address of "somewhere", which is of course not a valid location.
- You need to initialize a pointer by assigning it a valid address. This is normally done via the address-of operator (&).

Pointers

- … then, how do we get the value pointed by a pointer?
- But naturally… there is the *dereferencing operator* ***!

```
// Declare a pointer to int variables, declares an int and
sets the pointer
int *p1;  // pointer p1 is not initialized!!
int i=5;
p1=&i ;
i=*p1 ; // the dereferencing operator * applied to the
pointer p1 returns the value of the pointed to variable.
```


HOGESCHOOL ROTTERDAM

Dereference operator *

* operator is called **dereference** operator

```
int number = 88;
int *pNumber = &number;  // Declare and assign the address of variable number to
pointer pNumber (0x22ccec)
cout << pNumber<< endl;   // Print the content of the pointer variable, which
contain an address (0x22ccec)
cout << *pNumber << endl; // Print the value "pointed to" by the pointer, which is
an int (88)
*pNumber = 99;      // Assign a value to where the pointer is pointed to, NOT to the
pointer variable
cout << *pNumber << endl; // Print the new value "pointed to" by the pointer (99)
cout << number << endl;   // The value of variable number changes as well (99)
```

Address and dereferencing operator

```
p1=&v;
```

```
int *p1;`
```

```
int v=5;
```

```
int w=*p1;
```


HOGESCHOOL ROTTERDAM

Reference & vs Address Operator&
and
Pointer * vs Dereferencing Operator *

How to distinguish between the two semantic of &?

```
int &refNumber = number;
int *point2num
```

VS

```
int *point2num = &d;
int j = *point2num
```

IT'S NOT THE SAME!!!

HOGESCHOOL ROTTERDAM

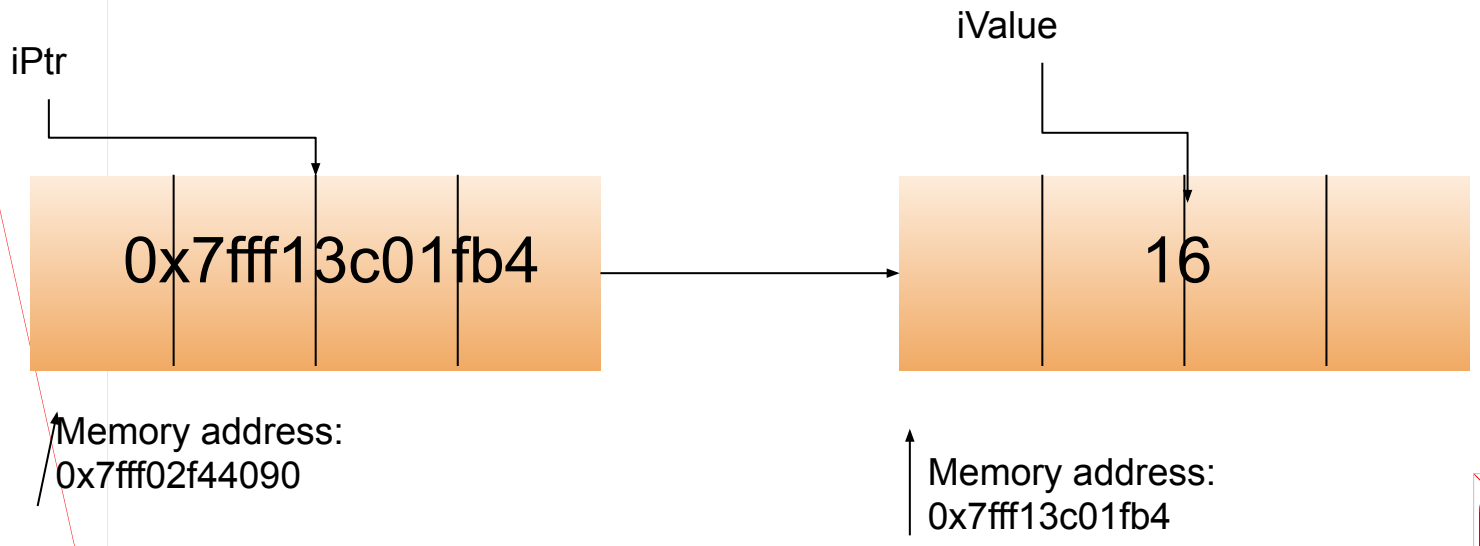
Reference &

- A reference is an *alias*, or an *alternate name* to an existing variable.
- The meaning of symbol `&` is different in an expression and in a declaration.
- If a datatype is present before the & then it is a reference otherwise it is an address operator

- It is used to provide *another name*, or *another reference*, or *alias* to an existing variable.

Pointers. A closer look...

```
// Declare a pointer variable called iPtr pointing to an int (an int pointer)
int iValue = 16;
int *iPtr; // It contains an address. That address holds an int value.
iPtr = &iValue;
```

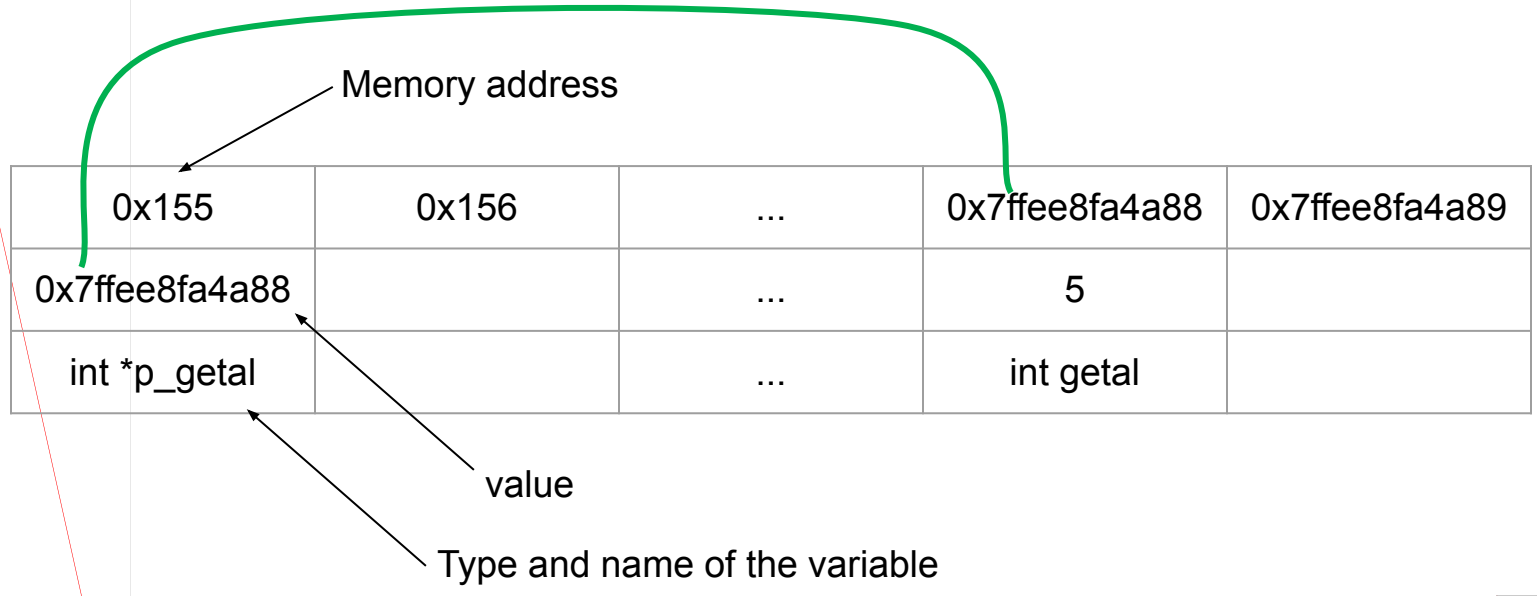


Pointers- In het geheugen

```
int getal = 5;
int *p_getal;
p_getal = &getal;
```

type int pointer

& address operator



Try it out

- Experiment with pointers: look what happens if you declare a pointer without initialisation (print it out)
- Initialize the pointer to an existing memory address and try out what happens

- Tips:
  - Use operators * and &
  - Try different variables types

Example

Pointers.cpp

Pointer has a Type!!

```
int i = 88;
double d = 55.66;
int *iPtr = &i;     // int pointer pointing to an int value
double *dPtr = &d; // double pointer pointing to a double value

iPtr = &d;    // ERROR, cannot hold address of different type
dPtr = &i;    // ERROR
iPtr = i;     // ERROR, pointer holds address of an int, NOT int
value

int j = 99;
iPtr = &j;  // You can change the address stored in a pointer
```

Try it out (15’ time)

- Declare an int variable and assign an initial value
- Declare a pointer variable pointing to an int (or int pointer) e.g. pNumber
- Assign the address of the variable number to pointer
- Print content of pNumber
- Print address of the int variable
- Print value pointed to by pNumber
- Print value of the int variable
- Re-assign value pointed to by pNumber
- Print content of pNumber
- Print address of the int variable
- Print value pointed to by pNumber
- Print value of the int variable
- Print the address of pointer variable pNumber

Solution

Example of solution

Pointers_1

Find the error

The following code fragment has a serious logical error!

```
int *iPtr;
*iPtr = 55;
cout << *iPtr << endl;
```

Null pointer

- You can initialize a pointer to 0 or `NULL`, i.e., it points to nothing.
- It is called a *null pointer*.

```
// Declare an int pointer, and initialize the pointer to point to nothing
int *iPtr = 0;
cout << *iPtr << endl;  // ERROR! STATUS_ACCESS_VIOLATION exception

int *p = NULL;          // Also declare a NULL pointer points to nothing
```

Reference &: example

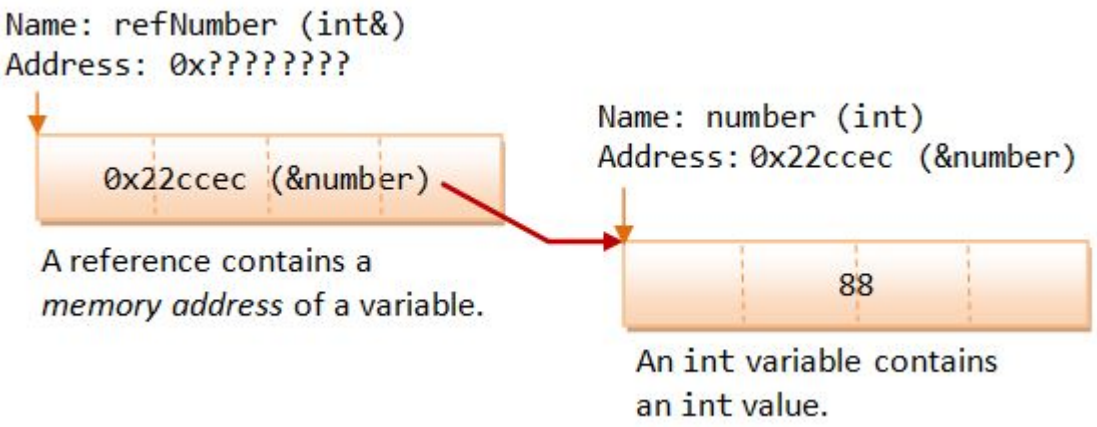
```
int main() {
    int number = 88;
    // Declare a reference (alias) to the variable number
    int &refNumber = number;
    // Both refNumber and number refer to the same value

    cout << number << endl;     // Print value of variable number (88)
    cout << refNumber << endl; // Print value of reference (88)
    refNumber = 99;             // Re-assign a new value to refNumber
    cout << refNumber << endl;
    cout << number << endl;     // Value of number also changes (99)

    number = 55;                // Re-assign a new value to number
    cout << number << endl;
    cout << refNumber << endl; // Value of refNumber also changes (55)
}
```

Reference: how does it work?

- A reference works as a pointer.
- A reference is declared as an alias of a variable.
- It stores the address of the variable



Right or wrong?

```
int c, *pc;
```

```
pc=c;
```
X

```
*pc=&c;
```
X

```
pc=&c;
```
✓

```
*pc=c;
```
✓

HOGESCHOOL ROTTERDAM

Pointers vs Reference

Pointers and references are equivalent, except:

- A reference is a *name constant for an address*. You need to initialize the reference **during declaration**.

```
int &iRef;    // Error: 'iRef' declared as
   reference but not initialized
```

- Once a reference is established to a variable, you cannot change the reference to reference another variable.

Pointers vs Reference

- To get the value pointed to by a pointer, you need to use the dereferencing operator `*` (e.g., if `pNumber` is a `int` pointer, `*pNumber` returns the value pointed to by `pNumber`).
- To assign an address of a variable into a pointer, you need to use the address-of operator `&` (e.g., `pNumber = &number`).
- On the other hand, referencing and dereferencing are done on the references implicitly

Example Pointers vs Reference

PointerVsReference.cpp

Question: what does this code fragment print out?

Assume a short is 2 bytes, and a 32-bit machine.

```
short value = 7; // &value == 0012FF60
short otherValue = 3;
// &otherValue == 0012FF54

short *ptr = &value;

std::cout << &value << '\n';\\ff60
std::cout << value << '\n';\\7
std::cout << ptr << '\n';\\ff60
std::cout << *ptr << '\n';\\7
std::cout << '\n';

*ptr = 9;

std::cout << &value << '\n';\\ff60
std::cout << value << '\n';\\9
std::cout << ptr << '\n';\\ff60
std::cout << *ptr  << '\n';\\9
std::cout << '\n';

ptr = &otherValue;

std::cout << &otherValue <<
    '\n';\\ff54
std::cout << otherValue << '\n';\\3
std::cout << ptr << '\n';\\ff54
std::cout << *ptr << '\n';\\3
std::cout << '\n';

std::cout << sizeof(ptr) << '\n';\\4
std::cout << sizeof(*ptr) << '\n';\\2
```

Question: what does this code fragment print out?

Assume a short is 2 bytes, and a 32-bit machine.

Solution

Try it out

- Write a program that displays the address of elements of an array using both array and pointers

- Tips:
  - Start for example with the following variables:
    - float arr[5];
    - float *ptr;
  - Arrays -> arrayA[i]
  - Pointers -> ….

HOGESCHOOL ROTTERDAM

Solution

ArrayPointers

Pointers and references in functions

- Reference are typically used in functions calls
- References are used as the function formal parameter for **pass-by-reference**
- Alternative to pass-by-reference is called **pass-by-value** (default in C++)

Pass-by-value: example

PassByValue

Pass-by-reference with pointer arguments

- Used when we want to modify the original copy directly (e.g. in passing huge object or array)
- avoid the overhead of cloning.

→ This can be done by passing a pointer of the object into the function, known as *pass-by-reference*.

Example

```
void square(int *);

int main() {
   int number = 8;
   cout <<  "In main(): " << &number << endl; // 0x22ff1c
   cout << number << endl;  // 8
   square(&number);         // Explicit referencing to pass an address
   cout << number << endl;  // 64
}

void square(int *pNumber) { // Function takes an int pointer (non-const)
   cout <<  "In square(): " << pNumber << endl; // 0x22ff1c
   *pNumber *= *pNumber;     // Explicit de-referencing to get the value
pointed-to
}
```

Try it out!

- Create a program which asks the user for 2 numbers
- Create a void function which takes an int* and int
- Pass the 2 numbers to the function and use a mathematical operator on them
- Print the value after the operation in the main

- Tips:
  - void …(int *, int)

Solution

Example of solution

Pointer_Multiply

Pass-by-reference with reference arguments

- Instead of passing pointers into function, you could also pass references into function, to avoid the clumsy syntax of referencing and dereferencing

→ This can be done by passing a pointer of the object into the function, known as *pass-by-reference*.

Example

```
void square(int &);

int main() {
   int number = 8;
   cout <<  "In main(): " << &number << endl; // 0x22ff1c
   cout << number << endl; // 8
   square(number);          // Implicit referencing (without '&')
   cout << number << endl; // 64
}

void square(int & rNumber) { // Function takes an int reference
(non-const)
   cout <<  "In square(): " << &rNumber << endl; // 0x22ff1c
   rNumber *= rNumber;        // Implicit de-referencing (without '*')
}
```

Try it out!

- Use the previous code and edit it to make the function take a reference
- Try to get the same result

- Tips:
  - void …(int &, int)

Solution

Example of solution

Reference_Multiply

Return value by pointer

- A C++ function can return a pointer.
- This pointer can, then, be dereferenced allowing access to the value of the pointed variable.
- The variable is then accessed by dereferencing (*)
- A function can be used on the left side of an assignment statement.

Example

```
#include <iostream>
#include <ctime>

using namespace std;

double vals[] = {10.1, 12.6, 33.1, 24.1, 50.0};

Double * setValues( int i ) {
   return (vals+i);   // return a pointer to the ith element
}

// main function to call above defined function.
int main () {
   *setValues(1) = 20.23; // change 2nd element
   *setValues(3) = 70.8;  // change 4th element
   return 0;
}
```

Return value by reference (alternative)

- A C++ function can return a reference in a similar way as it returns a pointer.
- When a function returns a reference, it returns an implicit pointer to its return value.
- A function can be used on the left side of an assignment statement.

Example

```
#include <iostream>
#include <ctime>

using namespace std;

double vals[] = {10.1, 12.6, 33.1, 24.1, 50.0};

double& setValues( int i ) {
   return vals[i];   // return a reference to the ith element
}

// main function to call above defined function.
int main () {
   setValues(1) = 20.23; // change 2nd element
   setValues(3) = 70.8;  // change 4th element
   return 0;
}
```

Be careful with reference returns!

- When returning a reference, be careful that the object being referred to does not go out of scope.
- It is not legal to return a reference to local var. But you can always return a reference on a static variable.

```
int& func() {
   int q;
   //! return q; // Compile time error
   static int x;
   return x;      // Safe, x lives outside this scope
}
```

L-values and R-values

- The term L-value is used for something that can appear on the left-hand side of an assignment operator.
- The term R-value is used for something that can appear on the right-handside of an assignment operator
- If you want the object return by a function to be an L-value, it must be returned by reference

Homework

Write three different programs to:

- Insert and display data entered by using pointer notation and arrays.
- Ask the user to enter elements of a matrix (of order r*c), computes the transpose of the matrix and displays it on the screen (with pointers and arrays).
- Ask the user to enter the size of the matrix (rows and columns). Then, it asks the user to enter the elements of two matrices and finally it multiplies two matrix and displays the result. To perform this task three functions are made:
  1. To take matrix elements from user
  2. To multiply two matrix
  3. To display the resultant matrix after multiplication

overtref jezelf