Pointers & Memory C++

Topics

1. Pointers and the Address Operator
2. Pointer Variables
3. Pointer Arithmetic
4. Initializing Pointers
5. Comparing Pointers
6. Pointers as Function Parameters
   a) **Pass-by-Value**
   b) **Pass-by-Reference with Pointer Arguments**
7. Constant Pointers

Pointers and the Address Operator

- Each variable in a program is stored at a unique address in memory
- Use the address operator **&** to get the address of a variable:

```
int num = -23;
cout << &num; // prints address
              // in hexadecimal
```

- The address of a memory location is a pointer

Pointer Variables

- Pointer variable (pointer): variable that holds an address
- Pointers provide an alternate way to access memory locations

Pointer Variables

- Definition:

```
int  *intptr;
```

- Read as:
  “**intptr** can hold the address of an int” or “the variable that **intptr** points to has type int”

- Spacing in definition does not matter:

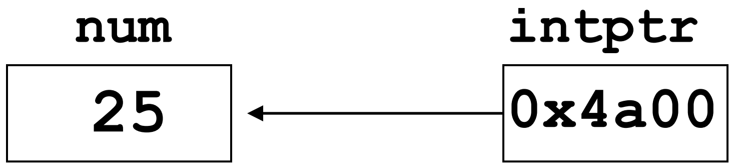
```
int * intptr;
int*  intptr;
```

Pointer Variables

- Assignment:

```
int num = 25;
int *intptr;
intptr = &num;
```

- Memory layout:



address of `num`: `0x4a00`

- Can access **`num`** using **`intptr`** and indirection operator **`*`**:

```
cout << intptr;  // prints 0x4a00
cout << *intptr; // prints 25
```

Pointer Arithmetic

Some arithmetic operators can be used with pointers:

- Increment and decrement operators **++**, **−−**
- Integers can be added to or subtracted from pointers using the operators **+**, **−**, **+=**, and **−=**
- One pointer can be subtracted from another by using the subtraction operator **−**

Pointer Arithmetic

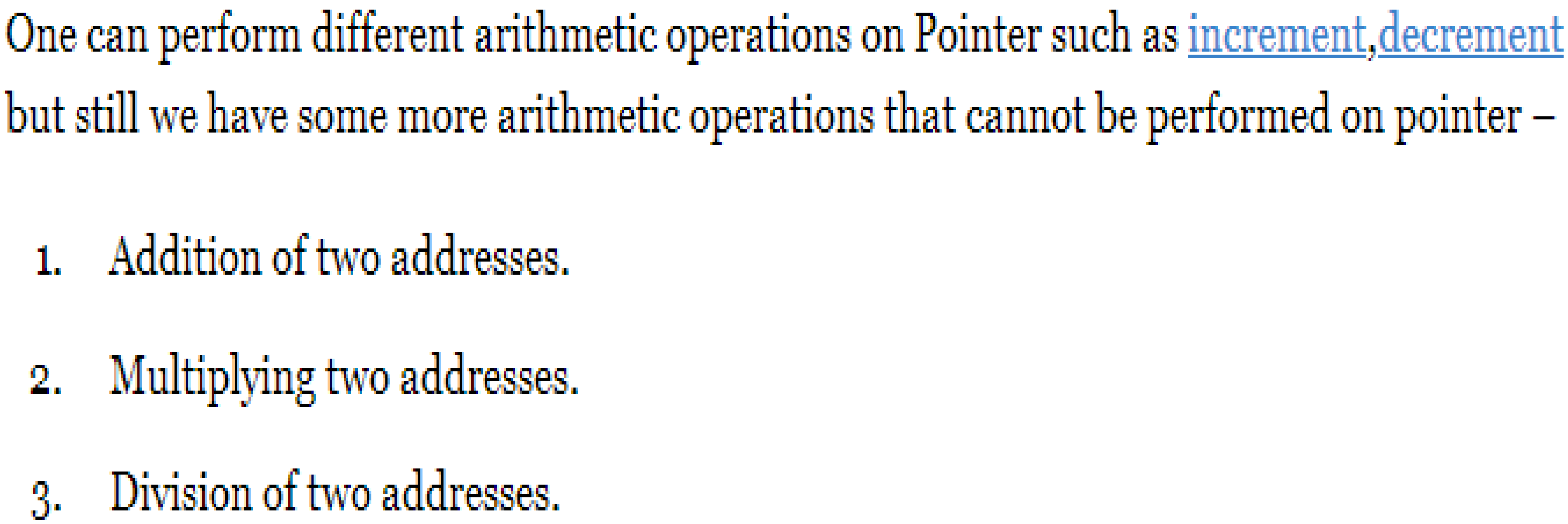
Assume the variable definitions

```
    int vals[]={4,7,11};
    int *valptr = &vals;
```

Examples of use of **++** and **--**

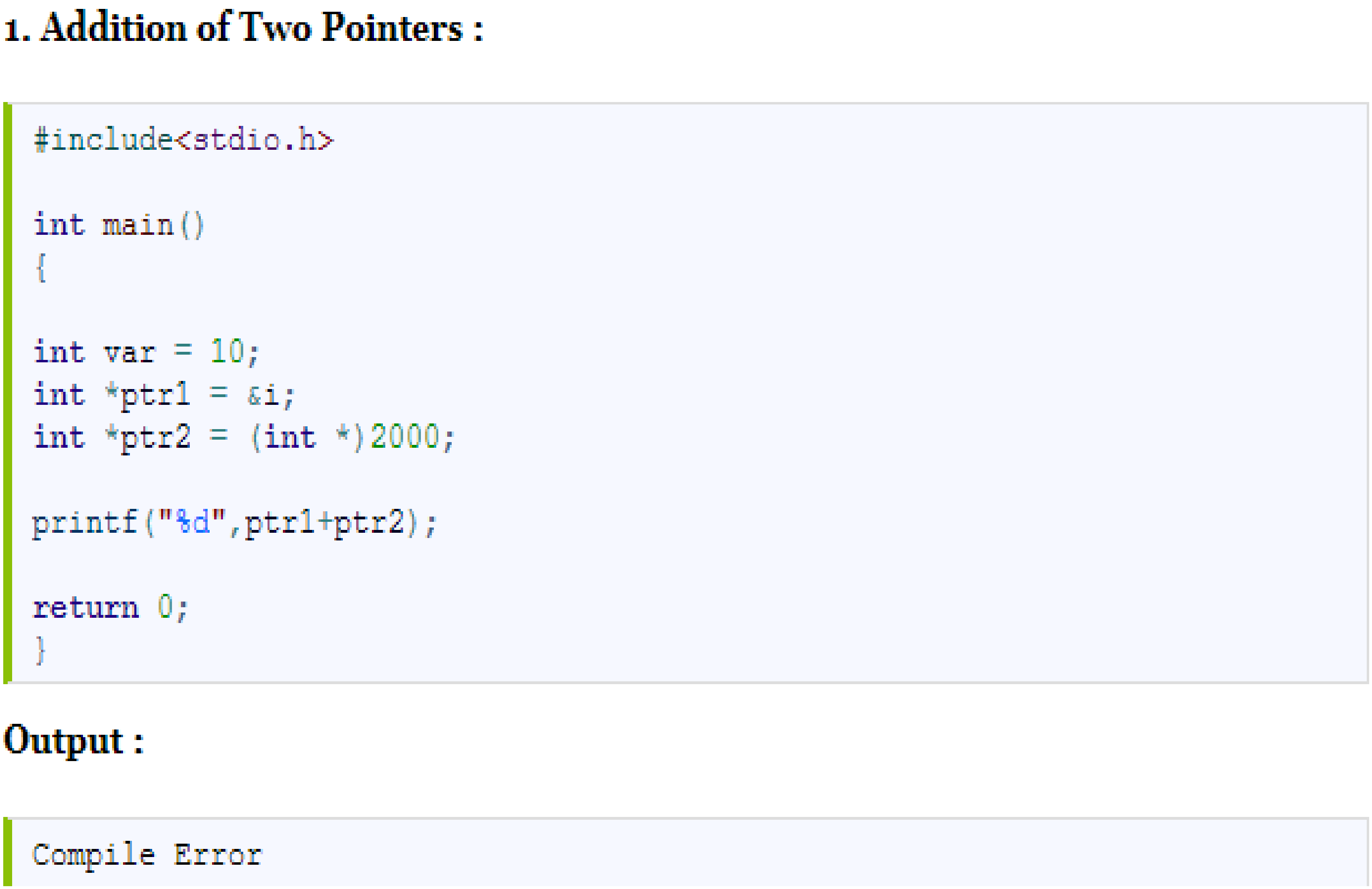
```
    valptr++; // points at 7
    valptr--; // now points at
  4
```

What is the invalid pointer arithmetic?



One can perform different arithmetic operations on Pointer such as increment,decrement but still we have some more arithmetic operations that cannot be performed on pointer –

1. Addition of two addresses.
2. Multiplying two addresses.
3. Division of two addresses.



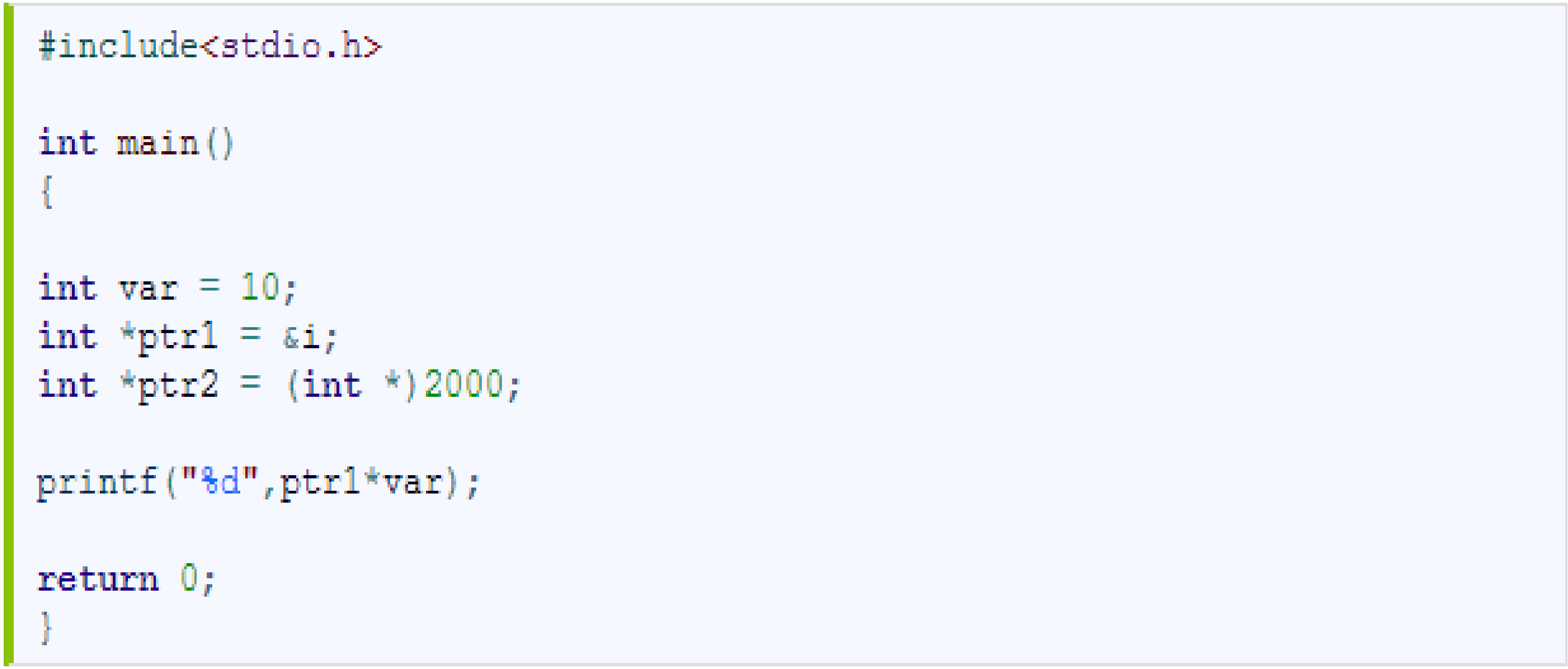
1. Addition of Two Pointers :

```
#include<stdio.h>

int main()
{

int var = 10;
int *ptr1 = &i;
int *ptr2 = (int *)2000;

printf("%d",ptr1+ptr2);

return 0;
}
```

Output :

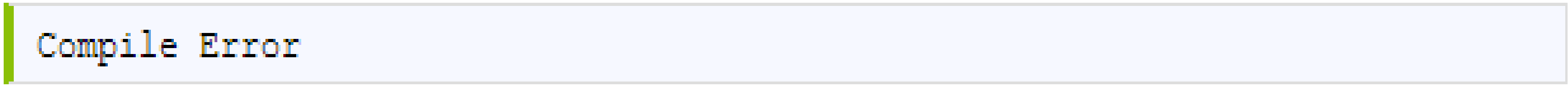
```
Compile Error
```

2. Multiplication of Pointer and number :



```
#include<stdio.h>

int main()
{

int var = 10;
int *ptr1 = &i;
int *ptr2 = (int *)2000;

printf("%d",ptr1*var);

return 0;
}
```

Output :



```
Compile Error
```

Initializing Pointers

- Can initialize to NULL or 0 (zero)

```
int *ptr = NULL;
```

- Can initialize to addresses of other variables

```
int num, *numPtr = &num;
```

- Initial value must have correct type

```
float cost;
int *ptr = &cost; // won't work
```

Comparing Pointers

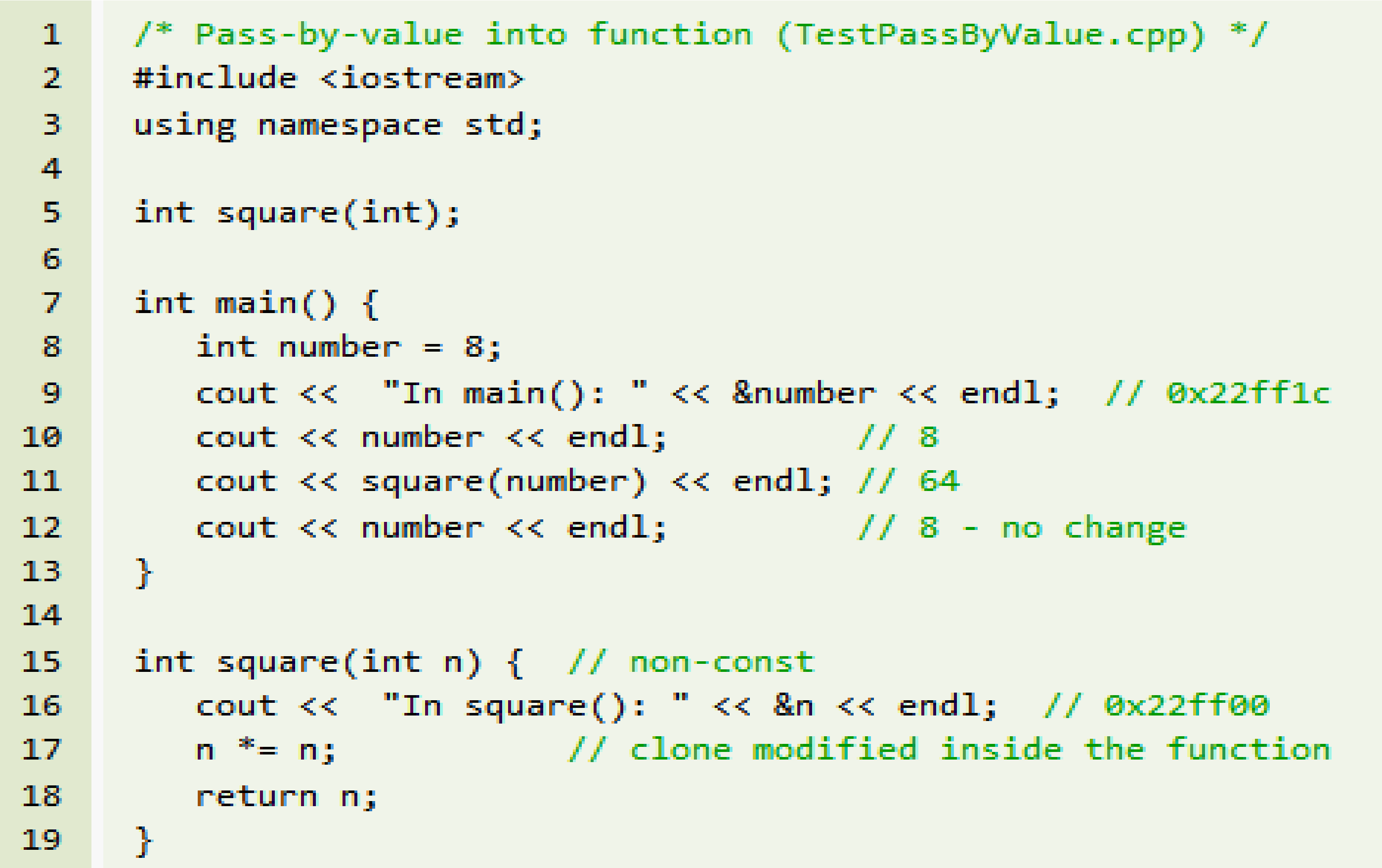
- Relational operators can be used to compare addresses in pointers
- Comparing addresses in pointers is not the same as comparing contents pointed at by pointers:

```
if (ptr1 == ptr2)   // compares
                    // addresses
if (*ptr1 == *ptr2) // compares
                   // contents
```

Pointers as Function Parameters

- **Pass-by-Value**
- In C/C++, by default, arguments are passed into functions *by value.*That is, a clone copy of the argument is made and passed into the function. Changes to the clone copy inside the function has no effect to the original argument in the caller. In other words, the called function has no access to the variables in the caller.

Pass-by-Value

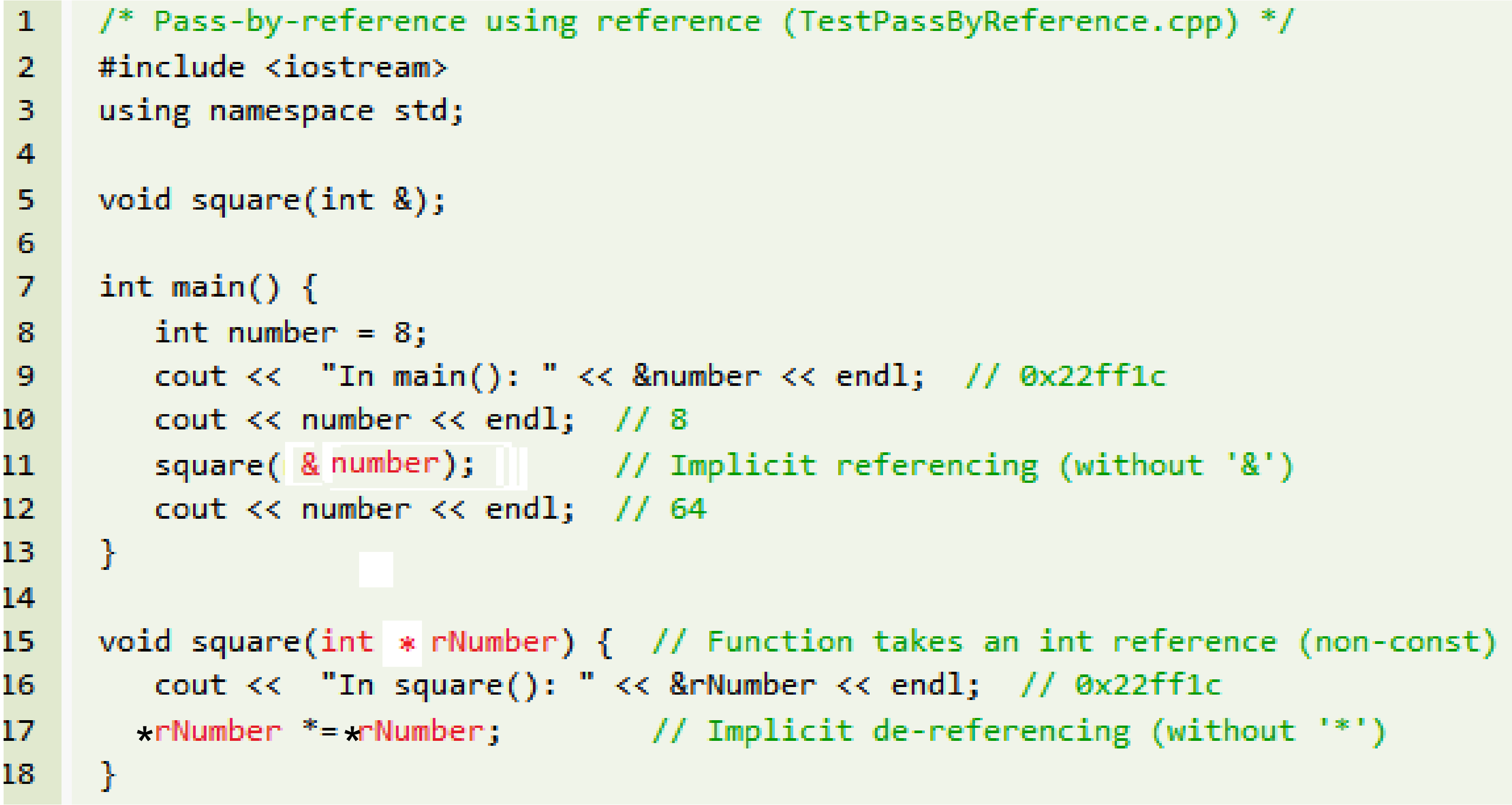


```cpp
/* Pass-by-value into function (TestPassByValue.cpp) */
#include <iostream>
using namespace std;

int square(int);

int main() {
   int number = 8;
   cout <<  "In main(): " << &number << endl;  // 0x22ff1c
   cout << number << endl;         // 8
   cout << square(number) << endl; // 64
   cout << number << endl;         // 8 - no change
}

int square(int n) {  // non-const
   cout <<  "In square(): " << &n << endl;  // 0x22ff00
   n *= n;           // clone modified inside the function
   return n;
}
```

Pointers as Function Parameters

- **Pass-by-Reference with Pointer Arguments**
- In many situations, we may wish to modify the original copy directly (especially in passing huge object or array) to avoid the overhead of cloning. This can be done by passing a pointer of the object into the function, known as *pass-by-reference.*

Pass-by-Reference with Reference Arguments



```
/* Pass-by-reference using reference (TestPassByReference.cpp) */
#include <iostream>
using namespace std;

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 '*')
}
```

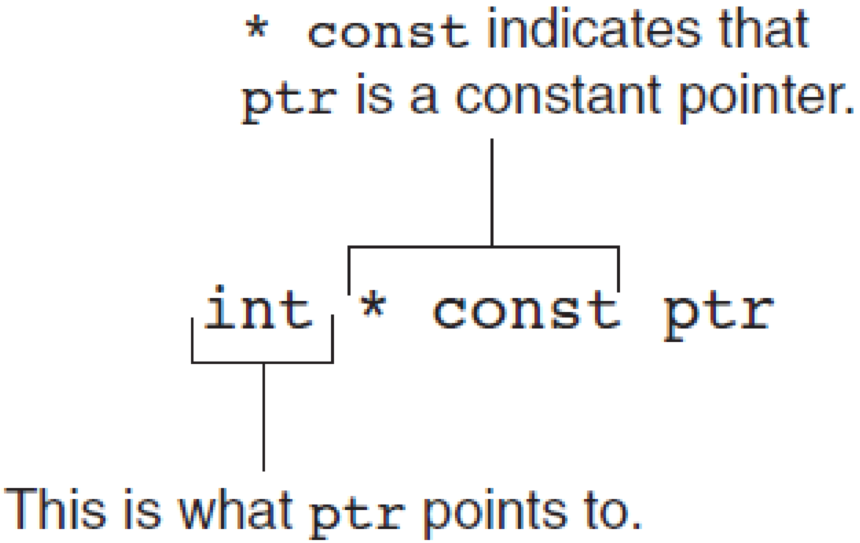
Constant Pointers

- Defined with **const** keyword adjacent to variable name:

```
int classSize = 24;
int * const classPtr = &classSize;
```

- Must be initialized when defined
- Can be used without initialization as a function parameter
  - Initialized by argument when function is called
  - Function can receive different arguments on different calls
- While the address in the pointer cannot change, the data at that address may be changed

Constant Pointer – What does the Definition Mean?



`* const` indicates that `ptr` is a constant pointer.

```
int * const ptr
```

This is what `ptr` points to.

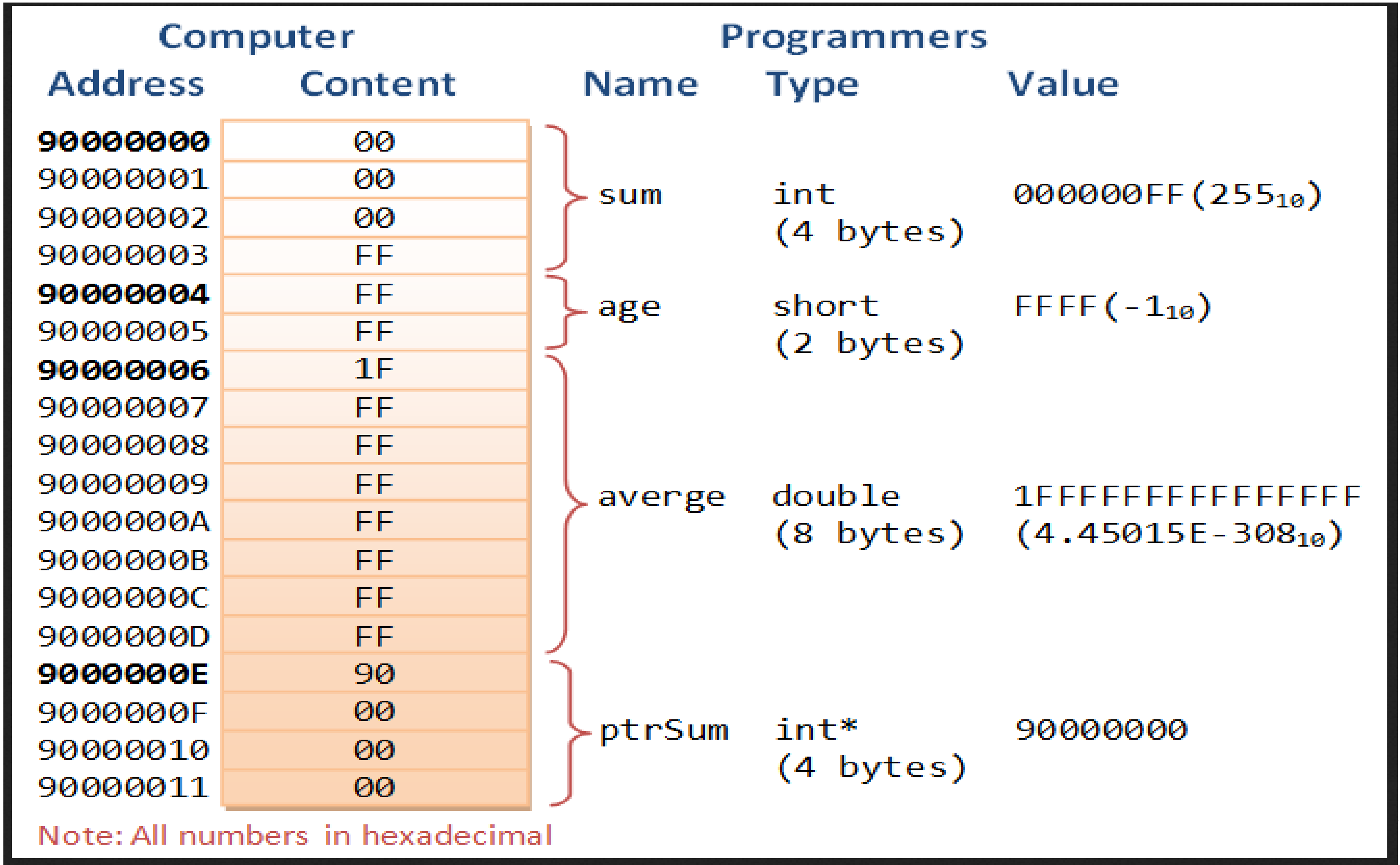
the data at that address may be changed

```
#include<stdio.h>
int main()
 {
 char c[20] = “SSUET";
char *const ptr = c;
*ptr= “Alighar “;
printf("%s\n", c);
return 0;
}
```

Pointer Variables

- Each address location typically hold 8-bit (i.e., 1-byte) of data. A 4-byte int value occupies 4 memory locations. A 32-bit system typically uses 32-bit addresses. To store a 32-bit address, 4 memory locations are required.

- The following diagram illustrate the relationship between computers' memory *address* and *content*; and variable's *name*, *type* and *value* used by the programmers.



Computer		Programmers		
Address	Content	Name	Type	Value
90000000	00	sum	int (4 bytes)	000000FF(255_{10})
90000001	00			
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