

C++ Data Types

- Fundamental data types
- User-defined data types
- Derived data types

C++ Fundamental Data Types

- Like C language
 - char, int, float, double, void
 - Modifiers can be applied
 - signed and unsigned - int and char
 - short - int
 - long - int and double
 - size and range of different data types depend on compiler and machine architecture
- Two new data types have been added to C++
 - bool and wchar_t

C++ Fundamental Data Types

- `bool`
 - `bool` variable can only hold *true* or *false*
 - Size of `bool` variables is implementation dependent
 - Default value of `bool` variables depends on storage class
 - `bool b1 = true; //true has value 1`
 - `bool b2 = false; //false has value 0`
 - *bool*, *true* and *false* are three new keywords
 - Guess the output: `bool b = 2.5; cout << b;`
 - *bool* variables and *true/false* keywords can be used in mathematical expression (e.g. `10 + b1 - false`)
 - In mathematical expression bools are elevated to `int`
 - Guess the output: `bool b = 2.5; int i = b; cout << i;`

C++ Fundamental Data Types

- `wchar_t`
 - Size is not fixed
 - Usually two bytes
 - Can be used to represent UTF-16 encoding
 - Four bytes on some platforms
 - You will rarely encounter this data type

C++ User-defined Data Types

- *struct* and *union*
 - Can be used as in C language
 - Many new features have been added for OOP. More about that when we start OOP
- New data type called *class* have been added
 - Major addition to C++ to enable OOP
 - More about that when we start OOP

C++ Derived Data Types

- Arrays
 - Similar to C language
 - One exception is related to initialization of character array
 - `char name[3] = "RAM";` //Valid in C, Error in C++
 - In C++, you must count for ending null character during static initialization of char array
- Functions
 - Major changes are introduced in C++
 - Many changes are driven by requirements for OOP
 - Will be discussed in later lectures

C++ Derived Data Types

- Pointers
 - Same as C language
 - Introduced pointer to constant Vs Constant pointer
 - We learned it in C. Not sure if it was adopted in C99 or was part of C89.

```
char name1[5] = "DDU";  
char name2[5] = "JNU";  
char * const ptr1 = name1; // ptr1[0] = 'P'; ptr1 = name2;  
char const * ptr2 = name1; // ptr2[0] = 'P'; ptr2 = name2;  
const char * ptr3 = name1; // ptr3[0] = 'P'; ptr3 = name2;  
char const * const ptr4 = name1; // ptr4[0] = 'P'; ptr4 = name2;
```

- Red color indicated invalid statement and green color indicates valid statement

C++ Derived Data Types

- Pointers

- Unlike C language, in C++ void pointer can't be assigned to non-void pointer. (**Why?**)

```
void *vptr; char *cptr;
```

```
cptr = vptr // valid in C, not in C++
```

```
cptr = (char *)vptr; // valid in C and C++
```

```
vptr = cptr; // valid in C and C++
```


C++ Derived Data Types

- References
 - New concept in C++
 - Creates an alias (alternate name) for a variable
 - Declaration is as follows:

```
data-type &reference-name = variable-name;  
int mumbai = 10;  
int &bombay = mumbai;
```
 - mumbai and bombay are referring to the same memory location, changing one will change other
 - Please note that neither mumbai nor bombay is pointer here
 - mumbai and bombay can be used interchangeably

C++ Derived Data Types

- References
 - Reference variable must be initialized at the time of declaration
 - Initialization of reference variable is completely different from assignment to it
 - There can be NULL pointer (pointer containing NULL address), but there can not be a NULL reference. Reference variable must refer to something, it can not refer to nothing.
- Unlike pointer, once initialized, reference variable can not be changed to refer to any other variable

C++ Derived Data Types

- References
 - Reference variable can be initialized with other reference variables of same type, but its not chaining like pointers
 - `int i; int &ir1 = i; int &ir2 = ir1;`
 - Addresses of i, ir1 and ir2 would be same in above case. And both ir1 and ir2 are int references
 - There can not be a reference to reference (no chaining)
- Array of references can not be created, but references can be created for arrays

```
int arr[10] = {1, 2};
int &ref = arr[5]; //ref is reference to one array element
int (&arr2)[10] = arr; //arr2 is reference to whole array arr
cout << arr2[0] << arr2[2]; //Guess the output
```

C++ Derived Data Types

- References
 - Pointer to reference is not allowed but reference to pointer is allowed. We will not discuss it in class to avoid confusion
 - For more information refer **this** link
 - It is possible to create reference to function (we will not look into it)
 - Guess the output:

```
int i = 7;  
int *ip = &i;  
int &ir = *ip;  
cout << ir;
```

 - Here, ir is a int reference, not pointer reference
 - `int i; int &ir = i; int *ip = &ir;`
 - Here type of ip is just an int pointer

C++ Derived Data Types

- References
 - `int i = 1;`
 - `const int j = 2;`
 - `int &ref1 = i;` // Valid
 - `// int &ref2 = j;` // Invalid
 - `const int &cref1 = i;` // Valid
 - `const int &cref2 = j;` // Valid
 - `ref1 = 5;` // Valid
 - `// cref1 = 5;` // Invalid
 - `// cref2 = 5;` // Invalid
- Constant pointer and pointer to const are separate concepts
- We can not change constant pointer to point to some other variable than what it has been initialized to point
- Pointer to const can not be used to alter value stored at location being pointed by it
- References are always constant by nature. So we will use **constant reference** and **reference to constant** interchangeably. Using **constant reference** we can not alter value being referred

C++ Derived Data Types

- References

- References can be created for temporary objects like literal constants, sum of two variables, return value of function etc.

const char &ref1 = 'A';

const int &ref2 = i + j; // where i and j are int variables

const float &ref3 = fun(); // where fun() returns float value

- Lifetime of temporary objects is tied to lifetime of its reference

- References can be created for user-defined data types too

struct s{int i;}s1; struct s &sr = s1;

C++ Derived Data Types

- References
 - Call by reference

```
#include<iostream>
using namespace std;
void fun(int &num)
{
    num++;
}
int main()
{
    int i = 10;
    fun(i);
    cout << i;
    return 0;
}
```

Output:
11

C++ Derived Data Types

- References

- Return by value

```
#include<iostream>
using namespace std;
int fun(int &num)
{
    num++;
    return num + 1;
}
int main()
{
    int i = 10;
    const int &ret_val = fun(i); //temporary object
    cout << i << " " << ret_val << endl;
    cout << &i << " " << &ret_val << endl;
    return 0;
}
```

Output:

11 12

0x7ffd566a1898 0x7ffd566a189c

C++ Derived Data Types

- References

- Return by value

```
#include<iostream>
using namespace std;
int fun(int &num)
{
    num++;
    return num;
}
int main()
{
    int i = 10;
    const int &ret_val = fun(i); //temporary object
    cout << i << " " << ret_val << endl;
    cout << &i << " " << &ret_val << endl;
    return 0;
}
```

Output:

11 11

0x7fff247b9b68 0x7fff247b9b6c

C++ Derived Data Types

- References

- Return by reference

```
#include<iostream>
using namespace std;
int &fun(int &num)
{
    cout << num << endl;
    num++;
    return num;
}
int main()
{
    int i = 10;
    int res = fun(i);
    int &ret_val = fun(i);
    fun(i) += 2;
    cout << i << " " << res << " " << ret_val << endl;
    cout << &i << " " << &res << " " << &ret_val << endl;
    return 0;
}
```

10

11

12

15 11 15

0x7fff78e63b28 0x7fff78e63b2c 0x7fff78e63b28

C++ Derived Data Types

- References

- Return by reference

```
#include<iostream>
using namespace std;
const int &fun(int &num)
{
    cout << num << endl;
    num++;
    return num + 2; // warning: returning reference to temporary
}
int main()
{
    int i = 10;
    const int &ret_val = fun(i);
    cout << i << " " << " " << ret_val << endl;
    cout << &i << " " << " " << &ret_val << endl;
    return 0;
}
```

- While returning by reference, developer must pay attention to scope of the variable/temporary whose reference is being returned.
- If temporary is being returned by the function then function return type must be constant, otherwise compiler will generate an error.

C++ Derived Data Types

- References

- References are limited in capability compared to pointers
 - But that makes references easy to use and simple to understand compared to pointers
 - Don't need to worry about NULL references
 - Can't be changed to refer to other variables once declared
 - No arrays of references
 - No chaining (No reference to reference)
 - No pointers to references
 - Anything that can be done using references can be achieved using pointers
- Use references when possible, use pointers when it is must
- Internally, most compilers implement references using pointers

C++ Data Types – type compatibility

- C++ is very strict compared to C when it comes to type compatibility
 - Necessary for function overloading
 - C++ does not treat character constants as integers
 - `sizeof(char)` and `sizeof('A')` is always 1 in C++ according to standard
 - In C, `sizeof('A')` is same as `sizeof(int)`

Interesting reads

- Fixed width integer types
 - <https://en.cppreference.com/w/cpp/types/integer>
- Size of bool is implementation specific
 - <https://stackoverflow.com/questions/4897844/is-sizeofbool-defined-in-the-c-language-standard>
- References FAQs
 - <https://isocpp.org/wiki/faq/references>