# 04 Review of C++ Basics

## 1 Variables

Built-in data types, e.g., int, double, etc.

Input and output, e.g., cin, cout.

Operators

- Arithmetic: +, -, *, etc.
- Comparison: <, >, ==, etc.
- x++ versus ++x

**Notes:**

**postfix(x++)**

```
x=0;
y=x++;//is equivalent to y=x;x=x+1;
//now x=1;y=0
```

**prefix(++x)**

```
x=0;
y=++x;//is equivalent to x=x+1;y=x;
//now x=1;y=1;
```

## 2 lvalue and rvalue

**lvalue:** An expression which may appear as **either the left-hand or right-hand side** of an assignment.

**Example:**

```
a=10;
b=a;
//a is lvalue;
```

**rvalue**: An expression which may appear on the right- but not left-hand side of an assignment.

**Example:**

```
10=a;//is illegal because 10 is rvalue
```

**Notes:**

- Any non-constant variable is an lvalue.
- Any constant is an rvalue.

# 3 Function

## Function declaration (or function prototype)

- Shows how the function is called.
- Must appear in the code before the function can be called.
- Syntax:

```
Return_Type Function_Name(Parameter_List);
//Comment describing what function does
```

**Example:**

```
int add(int a, int b);//Comment
```

**Tells:**

- **return type** (Type Signature)
- **how many arguments are needed** (Type Signature)
- **type of the arguments** (Type Signature)
- name of the function
- **formal parameter** names

## Function definition

- Describes how the function does its task.
- Syntax:

```
 Return_Type Function_Name(Parameter_List)
{

    //function code

}
```

**Example:**

```
int add(int a, int b)
{
    return a+b;
}
```

## Function Call Mechanisms

### Call-by-Value

**Example:**

```
void f(int x)
{
    x *=2;
}
```

**Call-by-Reference**

**Example:**

```
void f(int& x)
{
    x *=2;
}
```

Given the main function:

```
int main()
{
    …
    int a=4;
    f(a);
     …
}
```

**Notes:**

**Call-by-Value:** 4 in a ==> copy by value ==> 4 in x

x is local to function, a=4

**Call-by-Reference:** 4 in a ==> x is reference/alias to a ==> a=8

# 4 Array

An array is a **fixed-sized**, **indexed data type** that stores a collection of items, all of **the same type**.

**Example:**

```
int b[4];//declaration
b[0]=4;//assign value to array elements
int a=b[0];//access array elements
```

```
int sum(int a[], unsigned int size);
 // Returns the sum of the first
 // size elements of array a[]
```

**Notes:** Array is passed by **reference**.

**Example:**

```
void add_one(int a[], unsigned int size)
{
    unsigned int i;
    for (i=0; i<size; i++)
    {
        a[i]++;
    }
}

int main()
```

```
{
    int b[4];
    b[0]=1;b[1]=2;b[2]=3;b[3]=4;
    add_one(b,4);
    //now we have b[0]:2 b[1]:3 b[2]:4 b[3]:5
}
```

# 5 References

**Reference** is an alternative name for an object.

**Example:**

```
int ival = 1024;
int &refval = ival;
//refval is a reference to ival. We can change ival through refval.
```

**Notes:**

1. Reference **must be initialized** using a variable of the same type.
   **Reason:** When we declare a general variable (for example int a), C++ does the following steps **(1)** declare the variable type and name **(2)** allocate memory and address **(3)** initialize its value. **(2)** and **(3)** can be completed separately.
   However, when we decalre a reference type variable, C++ just bind the reference to the initial object in one step. Therefore we must initialize it with a lvalue.
2. There is **no way to rebind** a reference to a different object

```
int &refval2; // Error: not initialized
int &refval3 = 10; // Error: 10 is not a variable
```

```
int ival = 1024;
int &refval = ival;
int ival2 = 10;
refval = ival2;
// refval still binds to ival, not ival2.
// not rebinding, but refval and ival are both 10
```

# 6 Pointers

- They provide a convenient mechanism to work with arrays.
- They allow us to create structures (unlike arrays) whose size is not known in advance.

## Pointers and Arrays

**Example:**

```
int A[5]={1,2,3,4,5};
cout<<"A:"<<A<<endl;
int *p=A;
cout<<"P:"<<p<<endl;
cout<<"*P:"<<*p<<endl;
```

```
A:0x61fe00
P:0x61fe00
*P:1
```

**Notes:** A and P point to the address of A[0]. That is, A==&A[0], P==&A[0].

### Pointers and References

- A reference must be initialized with an object when it is declared. A pointer can be initialized at any time.
- Once a reference is initialized with an object, it cannot be changed to another object. A pointer can point to another object at any time.

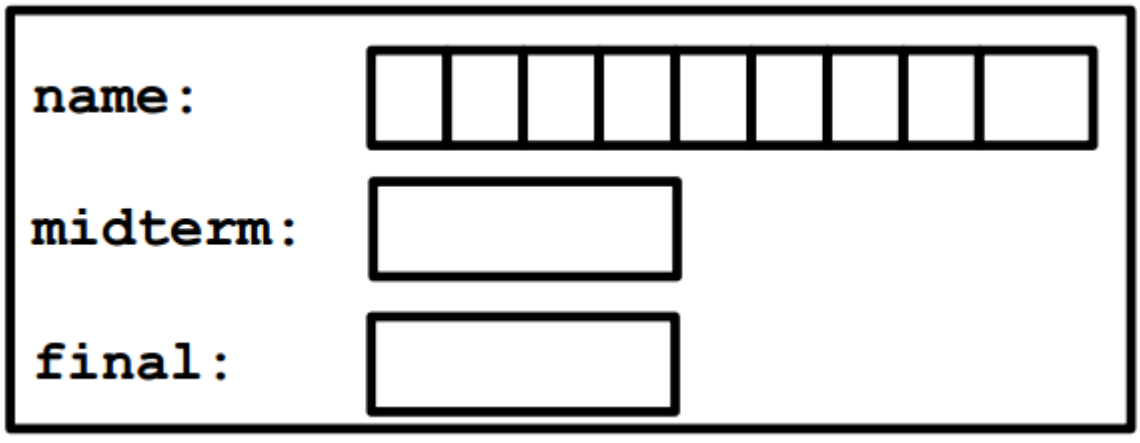
## 7 Structs

Declare a struct type that holds grades.

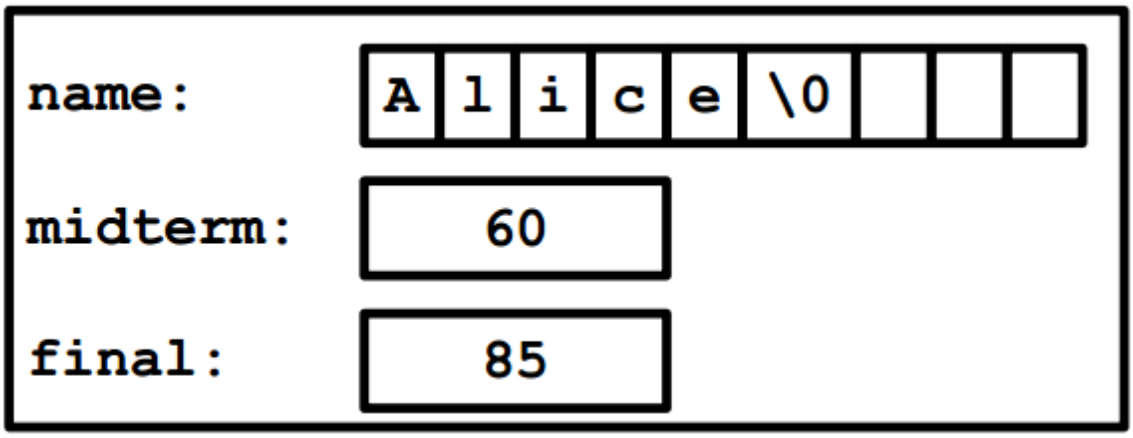
**Example:**

```
struct Grades // declare the type "struct grades",                // but does
not declare any objects of               // that type
{
    char name[9];
    int midterm;
    int final;
};

int main()
{
    ...
    struct Grades alice;// define single objects of                        //
this type
    ...
}
```




```
struct Grades alice= {“Alice”, 60, 85};
```




```
alice.midterm=65;// access its individual components                    //using
the “dot” operator

struct Grades *gPtr = &alice;
gPtr->final = 90;//a pointer to struct, visit component using "->"
```