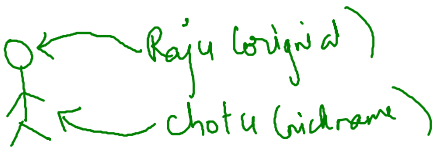
RV

{ khanam ✓
  299 ✓

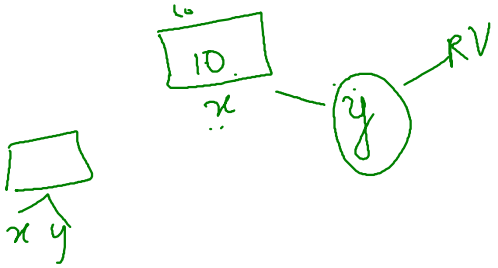


RV

int x = 10;

Syntax

int &y = x



int * ptr = &x;

same location

int a = 10;

int &ref = a; ref is a reference to 'a'

int& ref

int &ref

a++

a = 11

ref = 11

ref++

a = 12

ref = 12

int &y; ✗

int &y = x;

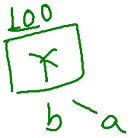
↳ valid variable;

int b = 1;

int &a = b;

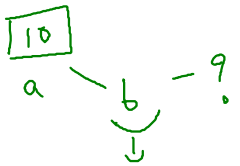
cout << &b //100

cout << &a //100



&b == &a

? RV



↓

function parameter by Reference.

-> Pass by reference.

```
void print(int x){
    x=x+1;
}
int main(){
    int x=90;
    cout<<"Before: "<<x<<endl;
  ✓ print(x);
   ✓cout<<"After: "<<x<<endl;
}
```

90 91

90
x

Pass by value

90
x

Before : 90
After : 90 //

-> Pass by pointer.

-> Pass by reference

```
print(int &x) {
    x = x + 1
}
int main() {
    int b = 9;
    Before << b ;
    print(b);
    After << b;
}
```

Pass by reference

9 10
b x

9
10

-> Return by reference

```
int& print(int &x) {
    return x ->
}
```

int i = 5;
int &y = i;

```
int main(){
    int a = 5;
    print(a);
}
```

```
int& print(int &b){
    return b;
}
```

b is reference of a

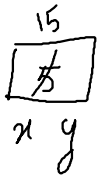
b is referring to original variable a.

Example

```
int main(){
    int x = 5;
    update(x);
    cout << x; //15
}
```

```
int& update(int &y){
    y = y + 10;   //5+10
    return y;
}
```

int &y = x



int &y = x

Types of scope

-> global scope.

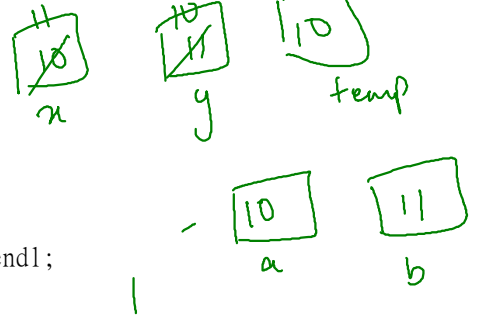
global variables X

RV

fun → RV

fun → fun

Pass by value



Before

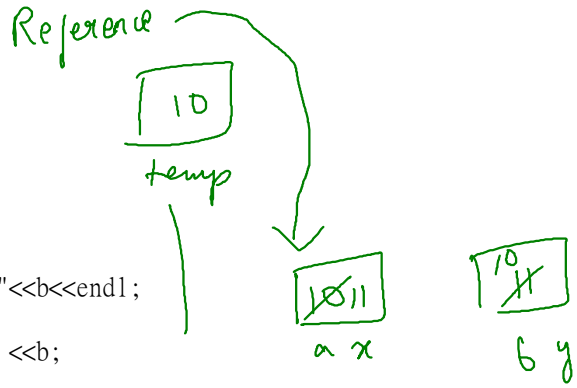
10 11

After

10 11

```
void swap(int x,int y){
    int temp=x;
    x=y;
    y=temp;
}
int main(){
    int a=10,b=11;
    cout<<"Before: "<<a<<" "<<b<<endl;
    swap(a,b);
    cout<<"After: "<<a<<" " <<b;
}
```

```
void swap(int &x,int &y){
    int temp=x;
    x=y;
    y=temp;
}
int main(){
    int a=10,b=11;
    cout<<"Before: "<<a<<" "<<b<<endl;
    swap(a,b);
    cout<<"After: "<<a<<" " <<b;
}
```



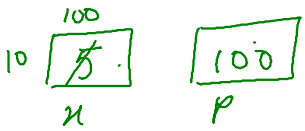
Before
10 11

After
11 10 //

RV , Pass by Reference .

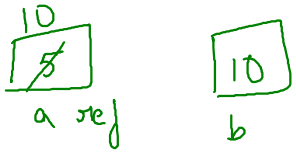
Q1.
```
int x = 10;
int &y = x;
y = 20;
cout << x;
```
A) 10 B) 20 C) 0 D) Error

Q2.
```
int x = 5;
int* p = &x;
*p = *p + 5;
cout << x;
```
A) 0 B) 5 C) 10 D) Garbage



Q3.
```
int& fun() {
  int a = 10;
  return a;
}
```
A) Safe B) Efficient C) Undefined D) Error

Q4.
```
int a = 5;
int &ref = a;
int b = 10;
ref = b;
cout << a;
```
A) 5 B) 10 C) Error D) Garbage

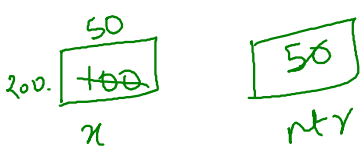


Q5.
```
int val = 30;
int& getVal() {
    return val;
}
int main() {
getVal() = 50;
    cout << val;
}
```
A) 30 B) 50 C) Error D) Garbage

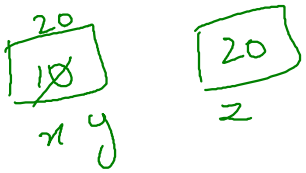
&val => 30
=> 50

Q6.
```
void update(int &x) {
      x += 10;
}
int main() {
    int a = 5;
    update(a);
     cout << a;
}
```
A) 5 B) 10 C) 15 D) Error

Q7.
```
int x = 100;
int *ptr = &x;
*ptr = 200;
cout << x;
```
A) 100 B) 200 C) 0 D) Error

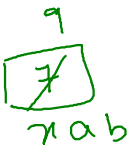


Q8.
```
int x = 10;
int &y = x;
int z = 20;
y = z;
cout << x;
```
A) 10 B) 20 C) Error D) 30



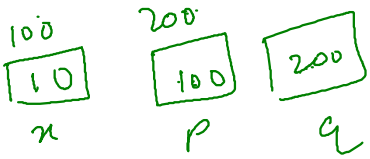
Q9.
```
int x = 10;
int &y = x;
y++;
cout << x;
```
A) 10 B) 11 C) 0 D) Error

Q10.
```
int x = 7;
int &a = x;
int &b = a;
b = 9;
cout << x;
```
A) 7 B) 9 C) Error D) Garbage



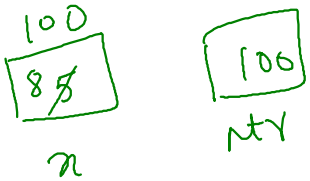
Q11.
```
int &ref;  // Assume outside function
```
A) Valid B) Needs initialization C) Will hold garbage D) Compiles fine

Q12.
```
int x = 10;
int *p = &x;
int **q = &p;
cout << **q;
```
A) 10 B) Address C) Error D) Garbage

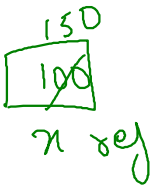


Q13.
```
int x = 10;
int *p = &x;
int y = *p + 5;
cout << y;
```
A) 10 B) 15 C) Error D) Garbage

Q14.
```
int x = 5;
int *ptr = &x;
*ptr += 3;
cout << *ptr;
```
A) 5 B) 8 C) 3 D) Error

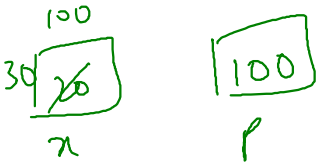


Q15.
```
int x = 100;
int &ref = x;
ref = ref + 50;
cout << x;
```
A) 100 B) 150 C) 50 D) Error

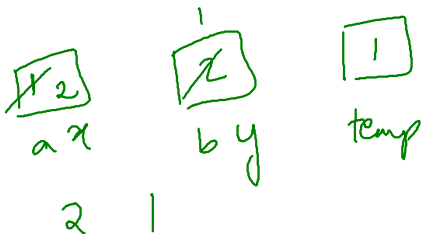


Q16.
```
int val = 1;
int& refFunc() { return val; }
int main() { refFunc()++; cout << val; }
```
A) 1 B) 2 C) Error D) Undefined

Q17.
```
int x = 20;
int *p = &x;
*p = 30;
cout << *p;
```
A) 20 B) 30 C) Error D) Undefined

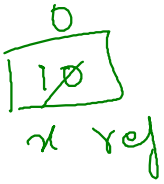


Q18.
```
int a = 1, b = 2;
void swap(int &x, int &y) {
int temp = x; x = y; y = temp;
}
int main() {
swap(a, b);
cout << a << " " << b;
}
```
A) 1 2 B) 2 1 C) Error D) Garbage



Q19.
```
int x = 10;
int &ref = x;
ref *= 2;
cout << x;
```
A) 10 B) 20 C) 5 D) Error

Q20.
```
int x = 10;
int &ref = x;
ref = 0;
cout << x;
```
A) 10 B) 0 C) Error D) Undefined



Homework

Q1. What does a reference variable store?
A) Value only
B) Address of another variable
C) Nothing
D) A copy of the variable

Q2. Choose the correct way to declare a reference variable:
A) int *ref = x;
B) int &ref = x;
C) int ref = &x;
D) ref int = x;

Q3. What is the output?
```
int x = 10;
int &y = x;
y = 20;
cout << x;
```
A) 10
B) 20
C) 0
D) Error

Q4. Which of the following is a valid reason to use references in function arguments?
A) To increase execution time
B) To reduce memory usage and modify original value
C) To hide variables
D) To convert types

Q5. What is the output?
```
int x = 5;
int* p = &x;
*p = *p + 5;
cout << x;
```
A) 0
B) 5
C) 10
D) Garbage

Q6. What is wrong in this code?
```
int& fun() {
    int a = 10;
    return a;
}
```
A) Nothing
B) Return type mismatch
C) Returning local reference
D) Syntax error

Q7. What is the result of returning a reference to a local variable?
A) Safe and efficient
B) Undefined behavior
C) Compilation error
D) Nothing happens

Q8. Choose correct usage of pass by reference in function:
A) void fun(int x)
B) void fun(int *x)
C) void fun(int &x)
D) Both B and C

Q9. What does ::x access?
A) Local variable
B) Reference variable
C) Global variable
D) Static variable

Q10. Global variables are initialized by default to:
A) Garbage value
B) 0
C) 1
D) NULL

Q11. Can you reassign a reference to another variable?
A) Yes
B) No
C) Only once
D) Depends on scope

Q12. What happens if reference is not initialized?
A) Reference gets default value
B) Compiler error
C) It is NULL
D) It behaves like pointer

Q13. What is the output?
```
int a = 5;
int& ref = a;
int b = 10;
ref = b;
cout << a;
```
A) 5
B) 10
C) Error
D) Garbage

Q14. Which memory section stores global variables?
A) Stack
B) Heap
C) Data segment
D) Code segment

Q15. What does this function do?
```
void fun(int &a) {
    a = a + 5;
}
```
A) Adds 5 to a copy of variable
B) Adds 5 to original variable
C) Does nothing
D) Error in syntax

Q16. What is the output?
```
int val = 30;
int& getVal() {
    return val;
}
int main() {
    getVal() = 50;
    cout << val;
}
```
A) 30
B) 50
C) Error
D) Garbage

Q17. What is the output?

```
int x = 5;
int *p = &x;
cout << *p;
```

A) 5
B) Address
C) Error
D) Garbage

Q18. Which is correct way to define a function that modifies the actual variable?
A) void change(int x)
B) void change(int *x)
C) void change(int &x)
D) Both B and C

Q19. Can a function return a reference?
A) Yes, always
B) No
C) Only if reference is to global or static variable
D) Only if returning pointer

Q20. What is the output?

```
int val = 100;
int& fun() {
    return val;
}
int main() {
    int x = fun();
    x = x + 50;
    cout << val;
}
```

A) 100
B) 150
C) 50
D) Error