fundamentals of programming I

Lab 6

Return Values

- If you want the function to return a value, you can use a data type(such as int ,string, etc.) instead of void, and use the return keyword inside the function:

Example

```
int myFunction(int x) {
  return 5 + x;
}

int main() {
  cout << myFunction(3);
  return 0;
}

// Outputs 8 (5 + 3)
```

Example

```
int myFunction(int x, int y) {
  return x + y;
}

int main() {
  cout << myFunction(5, 3);
  return 0;
}

// Outputs 8 (5 + 3)
```

Example

```
int myFunction(int x, int y) {
  return x + y;
}

int main() {
  int z = myFunction(5, 3);
  cout << z;
  return 0;
}
// Outputs 8 (5 + 3)
```

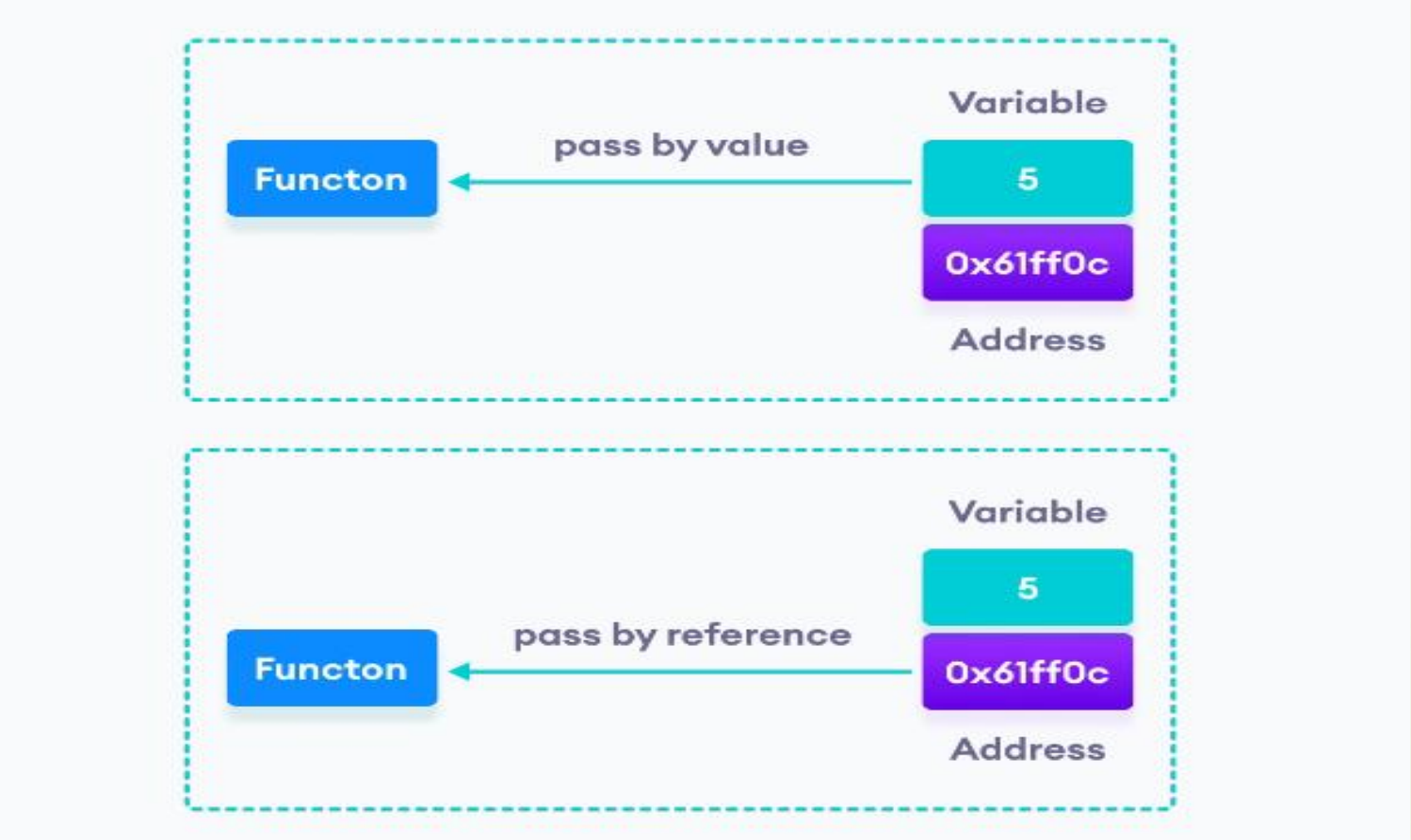
Pass By Reference

- In the examples from the previous page, we used normal variables when we passed parameters to a function. You can also pass a reference to the function so,we are using the address of the variable as our parameter.This can be useful when you need to change the value of the arguments.

Example

```
void swapNums(int &x, int &y) {
  int z = x;
  x = y;
  y = z;
}

int main() {
  int firstNum = 10;
  int secondNum = 20;

  cout << "Before swap: " << "\n";
  cout << firstNum << secondNum << "\n";

  // Call the function, which will change the values of firstNum and secondNum
  swapNums(firstNum, secondNum);

  cout << "After swap: " << "\n";
  cout << firstNum << secondNum << "\n";

  return 0;
}
```

Pass by Value vs. Pass by Reference



Pass Pointers as Funcation Parameters

```
#include <iostream>
using namespace std;

void swap(int* n1, int* n2) {
    int temp;
    temp = *n1;
    *n1 = *n2;
    *n2 = temp;
}

int main()
{
    int a = 1 , b = 3;
    int* ptr1=&a;
    int* ptr2=&b;
    cout << "Before swapping" << endl;
    cout << "a = " <<*ptr1 << endl;
    cout << "b = " <<*ptr2 << endl;

    // call function to swap numbers
    swap(ptr1, ptr2);

    cout << "\nAfter swapping" << endl;
    cout << "a = " << *ptr1 << endl;
    cout << "b = " << *ptr2 << endl;

    return 0;
}
```

Pass Arrays as Function Parameters

Example

```
void myFunction(int myNumbers[5]) {
  for (int i = 0; i < 5; i++) {
    cout << myNumbers[i] << "\n";
  }
}

int main() {
  int myNumbers[5] = {10, 20, 30, 40, 50};
  myFunction(myNumbers);
  return 0;
}
```

Example

```
#include <iostream>
using namespace std;
void enter_elements(int elements[],int s){
    for(int i=0;i<s;i++){
    cout<<"Enter element of array["<<i<<"]"<<endl;
    cin>>elements[i];
}
}
void print_elements(int print[],int c){
for(int i=0;i<c;i++)
    {
        cout<<"Elements of array are"<<endl;
        cout<<print[i]<<" ";
    }
}
int main(){
int size;
cout<<"Enter size of array"<<endl;
cin>>size;
int arr[size];
enter_elements(arr,size);
print_elements(&arr[0],size);

    return 0;
}
```

Function Overloading

- With **function overloading**, multiple functions can have the same name with different type of parameters or different number of parameters:

Example

```
int myFunction(int x)
float myFunction(float x)
double myFunction(double x, double y)
```

Overloading Using Different Types of Parameters

```
int plusFuncInt(int x, int y) {
  return x + y;
}

double plusFuncDouble(double x, double y) {
  return x + y;
}

int main() {
  int myNum1 = plusFuncInt(8, 5);
  double myNum2 = plusFuncDouble(4.3, 6.26);
  cout << "Int: " << myNum1 << "\n";
  cout << "Double: " << myNum2;
  return 0;
}
```

Overloading Using Different Number of Parameters

```
#include <iostream>
using namespace std;

void display(int var1, int var2) {
    cout << "First Integer number= " << var1;
    cout << " and second Integer number= " << var2 << endl;
}

void display(int var) {
    cout << "Integer number= " << var << endl;
}

int main() {

    int a = 5;
    int b = 10;

    display(a);

    display(a,b);

    return 0;
}
```

Thank You !