Week 3

Muhao Chen

Email: muhaochen@ucla.edu

Outline

- Function
- Break, continue, return

Function

A batch of statements: Input some parameters to the function, run a procedure, return a result.

Y=f(X)

A Simple Function

Type of the return value

```
int max(int a, int b) {
    if (a>b)return a;
    return b;
}
```

Declare parameters (using local variables)

```
int main(){
    int a;
    //call the function
    a=max(4,6); //a is 6
}
```

Another function

- //trim(). Remove all ‘ ‘ from the beginning and the end of a string.

```
string trim(string str) {
    string result=“”;
    int i,j;
    for (i=0; str[i]==‘ ‘ && i<str.size(); ++i);
    for (j=str.size() - 1; j>=i && str[j]==‘ ‘; --j);
    for (int k=i; k<=j; ++k)result+=str[k];
    return result;
}
```

```
return str.substr(i, j-i);
```

```
int main(){
    string s = “    Galneryus is a great band.      ”
    s = trim(s);
    cout<<s<<endl;  // “Galneryus is a great band.”
}
```

Void

- A function with a void return type: no return value. Also known as a procedure or subprogram in some other languages (e.g. pascal).

```
void printFactorial(int n) {
    int prod = 1;
    for (int i = 2; i <= n; i++)
            prod *= i;
    cout << "The factorial of " << n << " is " << prod << endl;
}
```

Void

- void f(…);
- int a; a=f();   X
- No return value!

Main()

- main() is also a function. The operating system calls main() to start the program.
- return 0 of main()
  - In some environment (e.g., a remote procedure call in a distributed system), the system need to know if a program ends successfully by returning a 0.

Everything inside a function is local

```
void printFactorial(int n) {
    int prod = 1;
    for (int i = 2; i <= n; i++)
            prod *= i;
    cout << "The factorial of " << n << " is " << prod << endl;
}

int main () {
    printFactorial(4);
    cout << prod; // compiler : prod is not defined
}
```

Everything inside a function is local

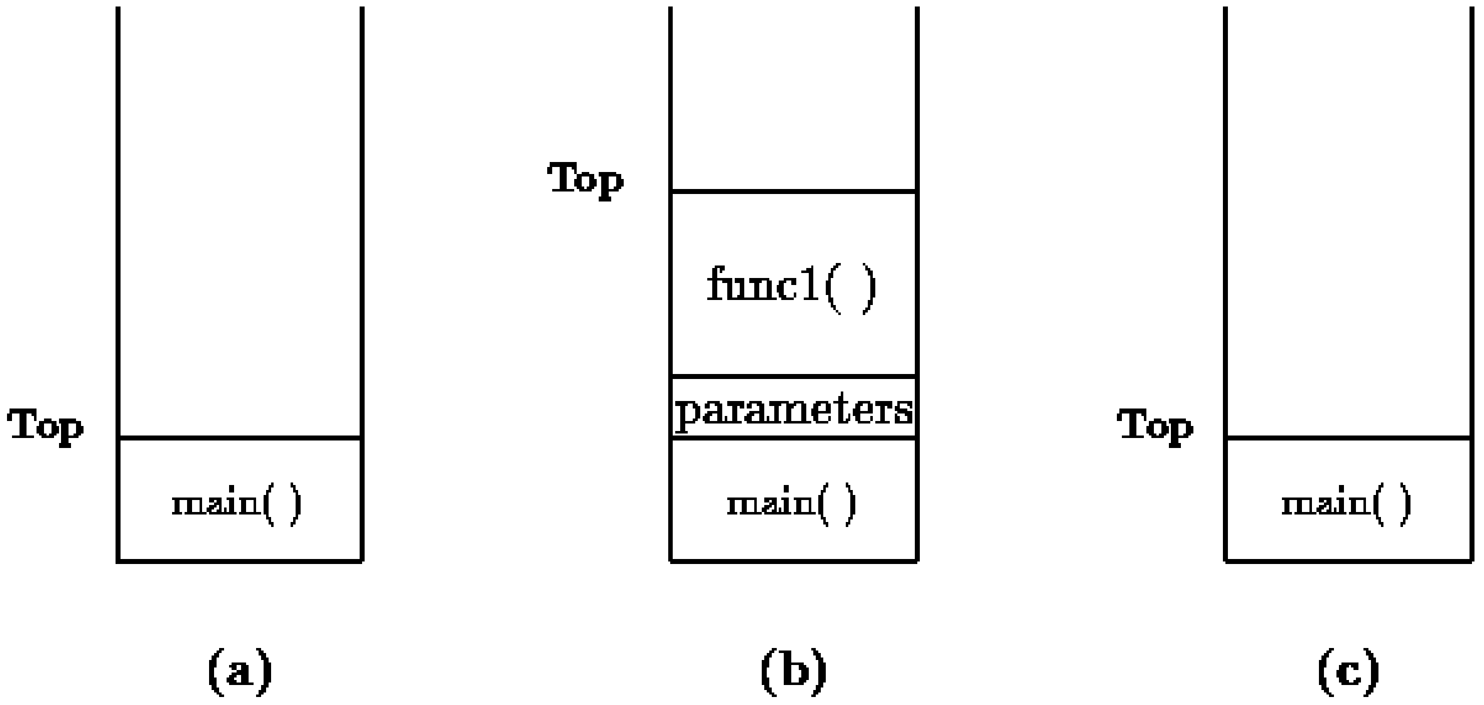


Figure 14.13: Organization of the Stack

Forbidden in functions

```
void function(int a, int b) {
	int a;
	double b;
	…
}
```

1. Define local variables using the same name as a parameter. (Compile error)   X

Forbidden in functions

```
bool non_negative(int a) {
        if (a>0)return true;
        else if(a<0)return false;
}
```

2. A condition (a==0) causes no return of a function. (undefined behavior; compile error in some IDE) X

Forbidden in functions

```
int function(int a, int b){
    double c = 1.0 * a / b;
    return c;
}
```

3. Inconsistent type of return value. X

Forbidden in functions

```
int function1() {
    int function2() {
            …
    }
    …
}
```

4. Nested definition of functions. X

Where to place the defined function

- Before the caller

```
int func(int a, int b){
    …
}

int main() {
    …
    c=func(4,6);
    …
}
```

- After the caller
  - Require signature

```
int func(int, int);

int main() {
    …
    c=func(4,6);
    …
}

int func(int a, int b){
    …
}
```

Can we define multiple functions with the same name

- Only if they have different combinations and/or types of parameters. (**Overloading**)

- int func(int a)
- int func(int a, int b)
- int func(int a, float b)
- int func(float c, int d)

Example

```
//func1
int max(int a, int b) {
    if(a>b)return a;
   return b;
}
```

```
//func2
float max(float a, float b) {
    if(a>b)return a;
   return b;
}
```

```
//func3
int max(int a, int b, int c) {
    //call func1
    return max(a, max(b, c));
}
```

We can always call a function in another function.

```
int main(){
     //call func1
     cout<<max(2, 3)<<endl;
     //call func2
     cout<<max(3.7, 4.0)<<endl;
     //call func3
     cout<<max(3,4,5)<<endl;
}
```

Can we define multiple functions with the same name

double func(int a, int b)

int func(int a, int b)

X It is impossible to tell which definition func(0, 1) corresponds to.

Signature of a function: name, #parameters, type of parameters. (Do not incl. type of return)

Formal Parameters & Actual Parameters

- **Formal parameters**: a.k.a. **Pass by Value** to a function, but won't change the values of the variables we use to pass the value. **(In fact they are just local variables.)**

```
int max(int a, int b) {
    if(a>b)return a;
   return b;
}

…
int x=4, y=6;
cout<<max(x,y);
```

max() uses the vals of x,y, but does not change their vals. It's OK.

```
void swap(int a, int b) {
    int tmp=a;
    a=b;
    b=tmp;
}
…
int x=4, y=6;
swap(x,y);
```

Change the vals of x,y? It won't work! X is still 4, y is still 6.

```
void mod(int a, intb) {
    a%=b;
}
…
int x=40, y=6;
mod(x,y);
```

Again, it won't work! X is still 40.

Actual Parameters (Pass by Reference)

- **Actual Parameters**: we want to both read and write the passed parameters.

```
void swap(int a, int b) {
    int tmp=a;
    a=b;
    c=a;
}
…
int x=4, y=6;
swap(x,y);
```

Changes to

```
void swap(int &a, int &b) {
    int tmp=a;
    a=b;
    c=a;
}
…
int x=4, y=6;
swap(x,y);
```

x is 6. y is 4

```
void mod(int a, int b) {
    a%=b;
}
…
int x=40, y=6;
mod(x,y);
```

Changes to

```
void mod(int &a, int b) {
    a%=b;
}
…
int x=40, y=6;
mod(x,y);
```

x is 40%6 -> 4

Example

```
void trim(string &str) {
    int i, j;
    for (i=0; str[i]==‘ ‘ && i<str.size(); ++i);
    for (j=str.size()-1; str[j]==‘ ‘ && j>=i; --j);
    str = str.substr(i, j-i);
    return;
}
```

Formal Parameters & Actual Parameters

- An easy way to remember when to use formal / reference parameters.

Type	Representation	R/W
Pass by Value	a	**Read-only**
Pass by Reference	**&a**	**Read-or-write**

Formal Parameters & Actual Parameters

- The principles
  - Pass by value: parameters are copied local variables
  - Pass by reference: no copying, pass the addresses

pass by reference
pass by value
cup =
cup =
fillCup( )
fillCup( )
www.penjee.com

void swap(int &a,int &b) { a=a^b; b=a^b; a=a^b; }

Silly, ordinary, and showy swaps (once on social media)

```
void swap(int a,int b)
{
    int temp;
    temp=a;
    a=b;
    b=temp;
}
```

Silly swap

```
void swap(int &a,int &b)
{
    int temp;
    temp=a;
    a=b;
    b=temp;
}
```

Ordinary swap

```
void swap(int &a,int &b)
{
    a=a^b;
    b=a^b;
    a=a^b;
}
```

Showy swap

An Open Question

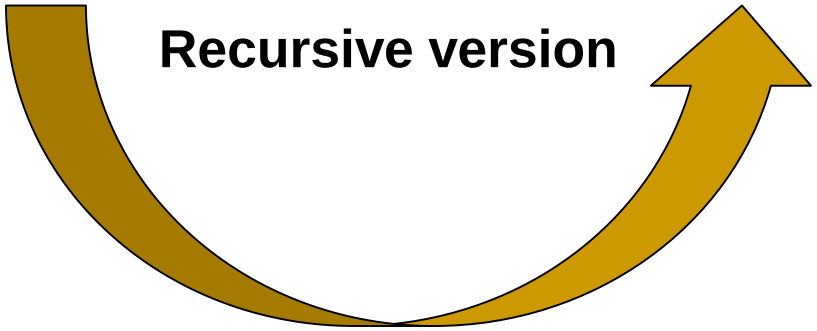
- During industrial development, people (almost) never use pass by value parameters in C++. (Why?)

Recursion

- A function that calls itself.

```
int factorial(unsigned int n) {
    result = 1;
    for (int i=2;i<=n;++i)result*=i;
    return result;
}
```

```
int factorial(unsigned int n) {
    if (n<=1) return 1;
    else return n * factorial(n-1);
}
```



Recursion

- More complex problems to solve with recursion:
  - Eight queen problem.
  - Hanoi tower.
  - Transitive closure.
  - Etc…
- We will see a lot of these in CS32.

Return; Break; Continue

Return; Break; Continue

- **Return**: terminates a function.
- **Break**: terminates a loop.
- **Continue**: terminates a cycle of a loop.

Return; Break; Continue

```
void main () {
    string s= “Nissan GTR”;
    for (int i=0;i<s.size();++i) {
        if(islower(s[i])) return;
        cout<s[i];
    }
    cout<<“ Nismo”;
}
```

```
void main () {
    string s= “Nissan GTR”;
    for (int i=0;i<s.size();++i) {
        if(islower(s[i])) break;
        cout<s[i];
    }
    cout<<“ Nismo”;
}
```

```
void main () {
    string s= “Nissan GTR”;
    for (int i=0;i<s.size();++i) {
      if(islower(s[i]))continue;
      cout<s[i];
     }
    cout<<“ Nismo”;
}
```

N	N Nismo	N GTR Nismo

Thank you!