Lecture 6

- Switch statement
- Functions
- Standard I/O
- Namespaces
- Variable Scope

Switch Statement

- Multi-way switch
- switch (<integer-expr>) {
  case <constant>:
  <statements>
  break;
  ...
  default:
  <statements>
  break;
  }

Beware of "falling through" if break is missing!

```
char c;
...
switch (c) {
  case '+':
    result = x + y;
    break;
  case '-':
    result = x - y;
    break;
  default:
    // all other
    // cases
    ...
    break;
}
```


Functions

- **Modular** programming
  - Breaking down tasks into **smaller sub-tasks**
- Increases **readability**
- Eases **debugging** and program **maintenance** because program pieces can be tested individually
- **Defining interfaces** makes programming in teams efficient

Function Examples

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

int square(int x) { return x*x; }

int fac(int n) { // n >= 0!
   if (n <= 1) return 1;
   return n*fac(n-1);
}

int gcd(int a, int b) { // a,b > 0!
  int r = a % b;
  if  (r != 0) return gcd(b, r);
  return b;
}

int bits_set(int x) {
  int n=0;
  while (x) { ++n; x &= x-1; }
  return n;
}
```

```
int main() {

  cout  << square(3)
        << ' '
        << gcd(3,5)
        << ' '
        << fak(3)
        << ' '
        << bits_set(7)
        << '\n';
   return 0;
}



> g++ test.c
> a.out
9 1 6 3
```

Function Declaration

```
int lcm(int a, int b);

void process_input();

double pow(double a, double b);
```

- Functions must be declared before they are used
- Syntax: <type> <function-name> (<param-list>); where

  <function-name> ::= <ident>
  <param-list> ::= ε | <list> (ε = empty word, | means or)
  <list> ::= <type> <ident> | <type><ident>, <list>
- return type **void** indicates that nothing is returned
- empty parameter list: no parameters are used

Function Definition

- Functions must be defined (possibly in a separate source file) if they are used
- Syntax: <type> <name> **(**<param-list>**)** { <statements> }
- Exit void functions with **`return;`**
- Values are returned by **`return`** <expr>**`;`**
  (type of expression must match function return type)
- Parameters are treated as local variables
- In C++, function definitions cannot be nested!

Some (Library) Functions

- int **main**(int argc, char *argv[ ]); - execution starts here
- void **exit**(int err); - exit execution with an error code (0 usually indicates success)
- int **abs**(int x); - compute the absolute value of x
- double **sin**(double x); - compute the sine of x
- double **floor**(double x); - round x down to nearest integer

To learn more about standard library functions consult the system header files located in /usr/include and $GCCHOME/include/c++ in conjunction with the UNIX man command (use "which g++" to find the gcc home directory).

E.g. less /usr/include/math.h
man floor

Standard Input & Output

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

int main() {
  int n;
  cout << "n=?\n";
  cin >> n;
  cout << "2*n=" << (2*n) << "\n";
  return 0;
}
```

- Input via input-stream **cin** ("standard input")
  - Syntax: **cin** >> <variable> >> ... >> <variable>;
- Output via output-stream **cout** ("standard output")
  - Syntax: **cout** << <expr> << ... << <expr>**;**
- **cin**/**cout** defined in standard header file <iostream>

Standard Error Stream

- Another predefined output stream: **cerr**
- Used for **error messages**
- Same output operator: <<
- Output is also sent to the console
- However, it is **not redirected** when using > or |
- Example:

```
cerr << "division by zero" << endl; exit(10);
```

On my web-page you can find a link to the iostream documentation which lists many useful functions. For instance cin.get() which reads one byte from the standard input (useful for assignment 1)

Two Example Files

- copy.c copy input to output
- calc.c add pairs of numbers
- in material/06

Namespaces

- Symbol collections which are qualified by name
  - types, variables, functions
- Avoids name conflicts
- **using namespace X**;
  - introduces all symbols of namespace X into the currenct context (no need for qualification)
  - e.g. **using namespace std;** -> introduces cin,cout,...
- **using X::y**;
  - symbol y is introduced as being an abbrevation for X::y
  - e.g. **using std::iostream;** -> introduces just iostream

- You can create your own namespace. E.g.
  - namespace **foo** { void bar(); }
  - call with: **foo**::bar() or
  - using namespace **foo**; bar();
- No namespace: symbols are put in global namespace (empty prefix)
  - e.g. ::strlen(s) // defined in <cstring> or <string.h>

Example

- Here is how to create your own namespace:

```
#include <iostream>

namespace My {
  int cout;
};

int main() {
  cout = 0;            // illegal: undeclared
  My::cout = 0;        // OK

  std::cout = 0;       // illegal: stream!
  std::cout << "foo";  // OK
}
```


Variable Scope

- Variables (and constants) have a **lifespan**
  - from the time they are created
  - until they are no longer used
- **Local variables** are declared within statement blocks enclosed by **{}**
- They are unknown outside the block
- Memory for them is allocated on the **system stack** and **not** automatically initialized
- When functions are exited, memory for local variables is released

Local Variable Scope Examples

```
int main()
{
  int   uninitialized;
  float initialized = 22.0/7.0;
  float x = 2.0; // (*)

  { // nested block
    float initialized = 3.1415; // (**)
    float x; // masks x (*)

    x = 2*initialized;  // refers to variable (**)
  }

  x = 3.1415926; // changes x (*)

  for (int i=10; i >= 0; --i) { cout << '?'; }
  i = 5; // i unknown here! local to for block

  int i; // variables can be defined anywhere!
  for (i=10; i >= 0; --i) { }
  // i lives here! value is -1
}
```


Memory Allocation in Functions

- Using a stack data structure
- Stackpointer (SP) points to next available byte in memory
- When a function is called the return address is first pushed onto the stack (e.g. store address at the location SP points to, add 4 to SP on 32 bit machines)
- Make room for local variables by increasing SP by a constant
- Upon function exit, decrease SP and jump to stored return address