

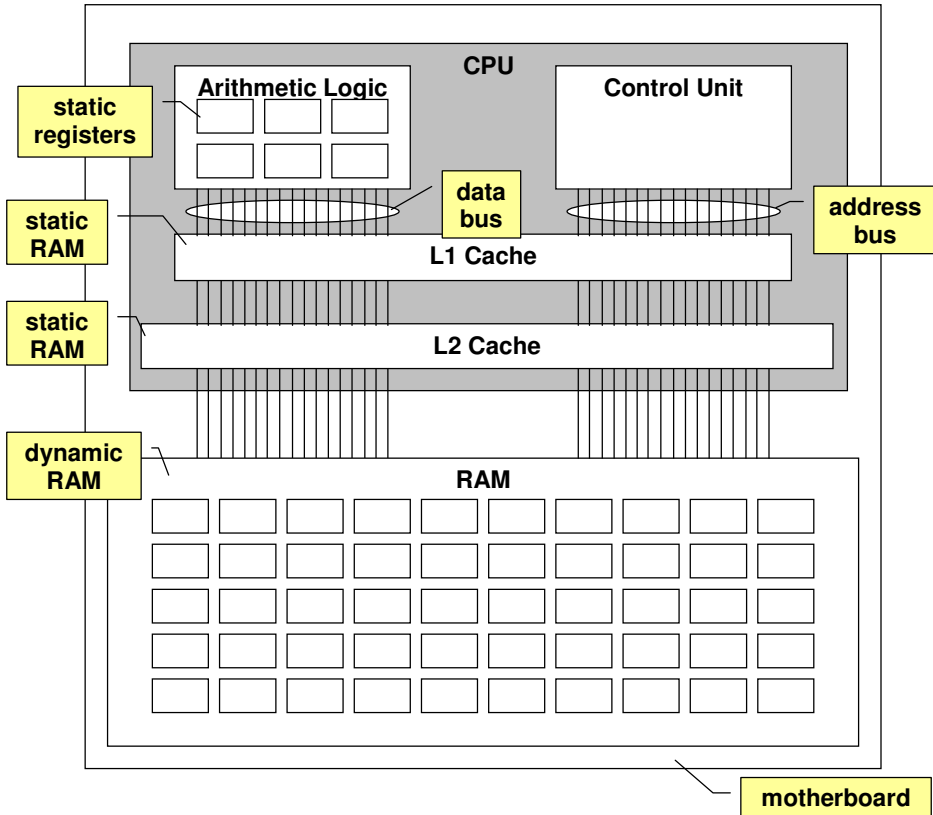
# Computer Studies in C++ Code Style and Compiler Optimization

## Guidelines

- define variables closest to use
  - code inherently more readable
  - *smart* compiler optimizes code
- use integer data types where possible
- Compiled Code Speed: Borland versus Microsoft

Debug code isolated from object code. Raw executables always fast.

Debug code embedded in object code. Must build a release version.



```
void SelectionSort(int* panNumbers, int nSize)
{
  int nLastIndexToSort = nSize - 1;
  int j;
  int nTemp;
  int nUnsorted;
  int nIndexToSmallest;

  for (nUnsorted = 0; nUnsorted < nLastIndexToSort; nUnsorted++) {

    nIndexToSmallest = nUnsorted;
    for (j = nIndexToSmallest + 1; j < nSize; j++) {
      if (panNumbers[j] < panNumbers[nIndexToSmallest])
        nIndexToSmallest = j;
    }

    nTemp = panNumbers[nIndexToSmallest];
    panNumbers[nIndexToSmallest] = panNumbers[nUnsorted];
    panNumbers[nUnsorted] = nTemp;
  }
}
```

**Traditional C Style:** variable definitions at top of function

```
void SelectionSort(int* panNumbers, int nSize)
{
  int nLastIndexToSort = nSize - 1;
  for (int nUnsorted = 0; nUnsorted < nLastIndexToSort; nUnsorted++) {

    int nIndexToSmallest = nUnsorted;
    for (int j = nIndexToSmallest + 1; j < nSize; j++) {
      if (panNumbers[j] < panNumbers[nIndexToSmallest])
        nIndexToSmallest = j;
    }

    int nTemp = panNumbers[nIndexToSmallest];
    panNumbers[nIndexToSmallest] = panNumbers[nUnsorted];
    panNumbers[nUnsorted] = nTemp;
  }
}
```

**C++ Style:** variable definitions closest to use

## Guidelines for Inline Functions

- *inline* function syntax is a polite request to create *inline* function: may or may not be honoured by the compiler.
- always use for simple *Get()* functions: essentially create *read-only* variables.
- never use for large functions (a dozen lines or more).
- Microsoft does not treat as *inline* in *Debug* version, only in *Release* version.
- Borland provides compiler switch to turn determine *inline* function handling.

```
class Employee {
private:
    int nEmployeeNumber;
    char* pszName;
    // more data members
public:
    // more member functions
    int GetEmployeeNumber() { return nEmployeeNumber; }
    // more member functions
};

void main()
{
    const int nMAX_STAFF = 10;
    Employee aeStaff[nMAX_STAFF];
    // more code to input data

    // display all Employee Numbers
    for (int i = 0; i < nMAX_STAFF; i++)
        cout << aeStaff[i].GetEmployeeNumber() << endl;
}
```

Function declared and defined inside the class is *inline* function.

Function call syntax, but machine code not implemented as a function call. Direct memory access embedded at the machine code level.

## Implications

- Speed:
- Code Size: