File Structures

04.B. Fundamental File Structure Concepts

2020. Spring

Instructor: Joonho Kwon

jhkwon@pusan.ac.kr

Data Science Lab @ PNU

Slides are based on the textbook.

data science laboratory

Outline

- 4.1 Field and Record Organization
- 4.2 Using Classes to Manipulate Buffers
  - Buffer class for delimited fields
  - Buffer class for length-based fields
  - Buffer class for fixed-length fields
- 4.3 Using Inheritance for Record Buffer Classes
- 4.4 Managing Fixed-Length, Fixed-Field Buffers
- 4.5 An Object-Oriented Class for Record Files

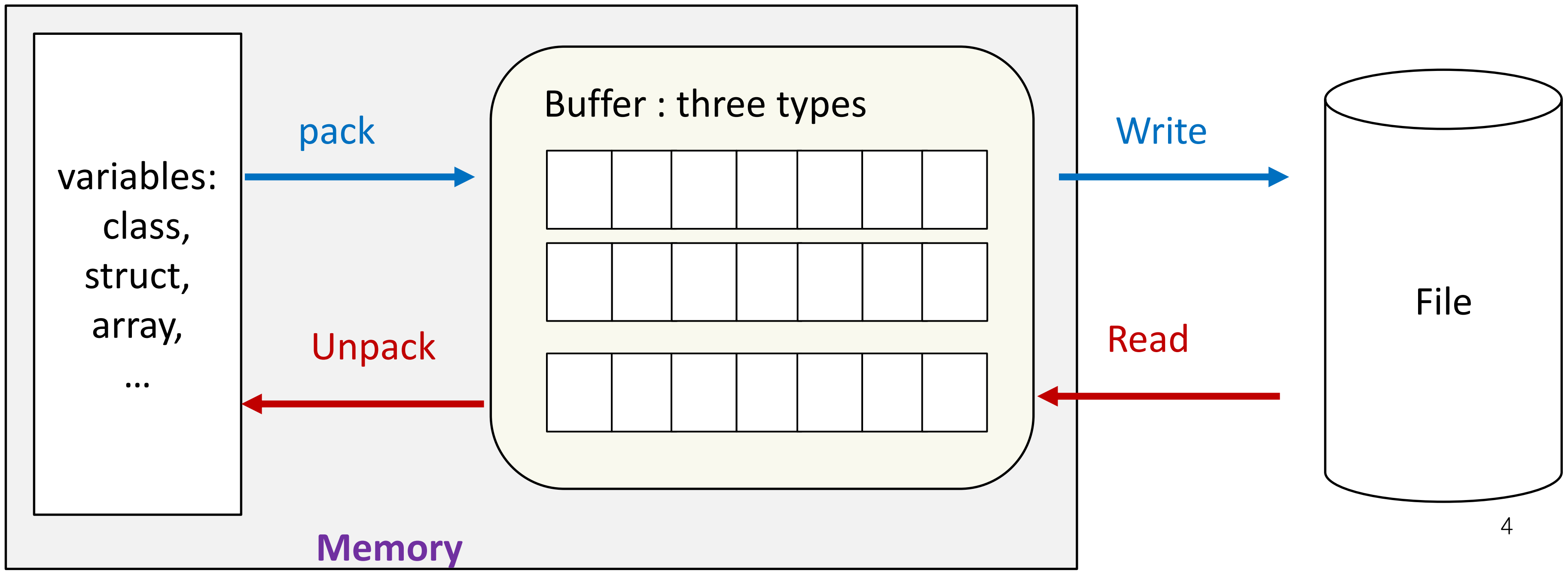
Detour: Buffer (from ch03.B)

- Definition
  - the part of main memory available for storage of copies of disk blocks
- Program buffers vs. System I/O buffers

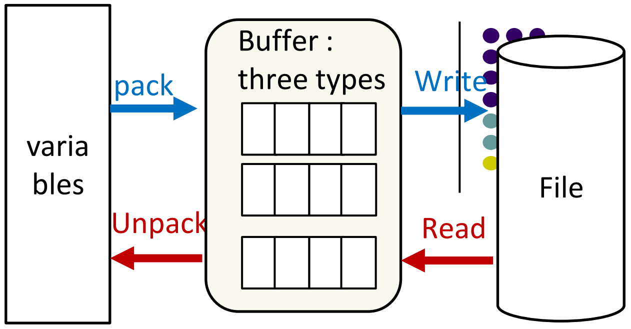
- Buffer manager
  - subsystem responsible for the allocation for blocks
  - goal:
    - minimize the number of disk access
    - utilize the memory space effectively

I/O interfaces for Buffers

- How to do
  - Pack from memory according to the characteristics of Buffer and write to file
  - On the contrary, read and unpack.



Use classes for buffers (1/2)

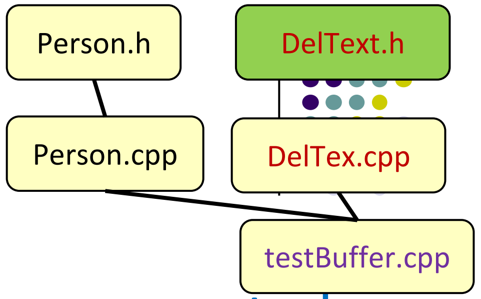


- C++ classes for buffer objects
  - encapsulate the pack( ), unpack( ), read( ), and write( ) operations of buffer objects
  - buffer classes for output
    - start with an empty buffer objects
    - pack( ) field values into the buffer object one by one
    - write( ) the buffer contents to an output stream
  - buffer classes for input
    - initialize a buffer object
    - read( ) a record from an input stream
    - unpack( ) the object's field values, one by one

Use classes for buffers (2/2)

- Three classes for buffers
  - Buffer class for delimited fields
  - Buffer class for length-based fields
  - Buffer class for fixed-length fields

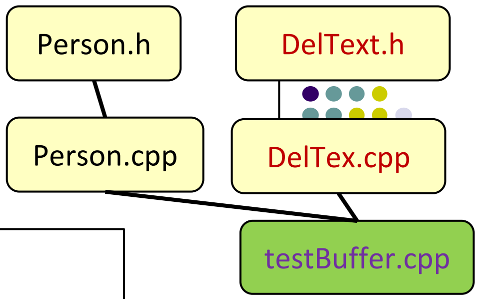
Buffer class for delimited fields



- support variable-length buffers whose fields are represented as delimited text
  - in file deltext.h in Appendix E, the page 595
  - Where to define Read(), Write(), Pack(), Unpack() : file stream ? Class Buffer? Class Person?

```
class DelimTextBuffer{
public:
    // construct with fiedls with delimeters
    DelimTextBuffer (char Delim = '|', int maxBytes = 1000);
    void Clear();       // clear fields from buffer
    int Read (istream & file);
    int Write (ostream & file) const;
    int Pack (const char * str, int size = -1);
    int Unpack (char *str);
    int Init (char delim, int maxBytes = 1000);
private:
    char Delim;         //delimiter character
    char DelimStr[2];   // zeoro terminated string for Delim
    char * Buffer;      // character array to hold field values
    int BufferSize;     // current size of packed fields
    int MaxBytes;       // max # of characters in buffer
    int NextByte;       //packing, unpacking position in buffer
};
```

How to use DelimTextBuffer



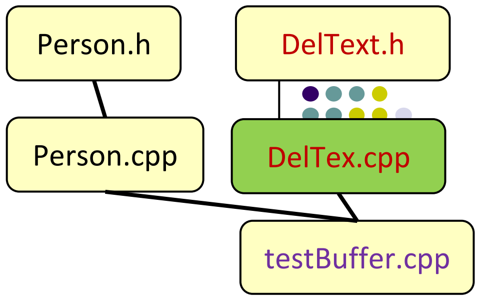
```
Person maryAmes;
DelimTextBuffer buffer; //FixedFieldBuffer buffer;
buffer.pack(MaryAmes.lastName);
buffer.pack(MaryAmes.FirstName);
...
buffer.pack (MaryAmes.zipcode);
buffer.write(stream);
```

- Usage
  - declare objects of class Person and class DelimTextBuffer
  - pack the person into the buffer
  - write the buffer to a file

- the declaration of object buffer

```
DelimTextBuffer buffer; //default arguments
DelimTextBuffer buffer('|', 1000)
```

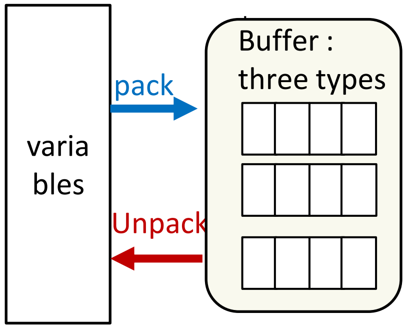
DelTextBuffer Initialize



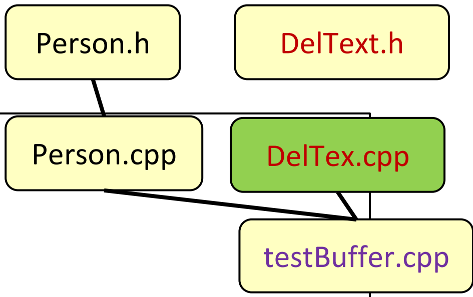
```
int DelimTextBuffer :: Init (char delim, int maxBytes)
 // construct with a maximum of maxFields
{
    Delim = delim;
    DelimStr[0] = Delim;
    DelimStr[1] = 0;
    if (maxBytes < 0) maxBytes = 0;
    MaxBytes = maxBytes;
    Buffer = new char[MaxBytes];
    BufferSize = 0;
    return 1;
}

DelimTextBuffer::DelimTextBuffer(char delim, int maxBytes)
// constgruct with a maximum of maxFiedls
{
    Init(delim, maxBytes);
}
```

Pack() (1/2)

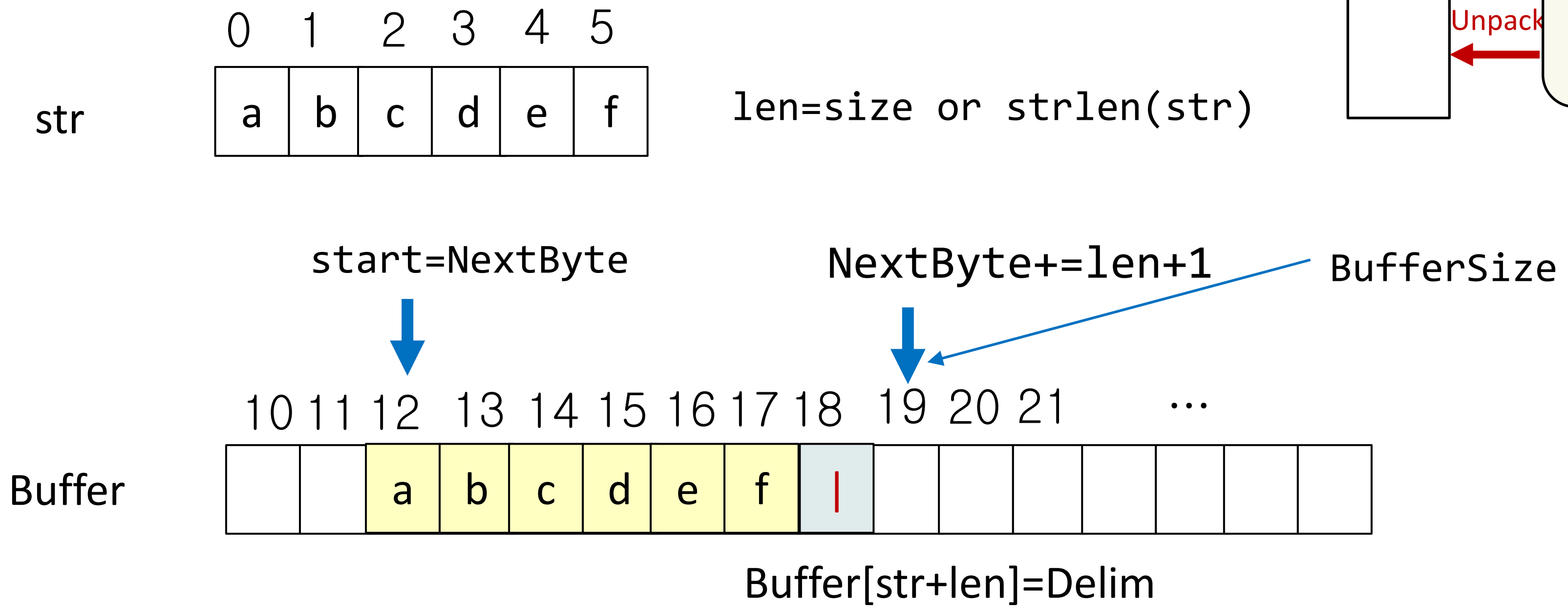


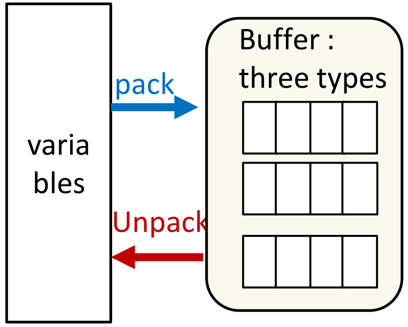
- the pack method for fields
  - copy the characters of str (memory) into the buffer
  - add the delimiter character



```
int DelimTextBuffer :: Pack (const char * str, int size)
{
    short len; // length of string to be packed
    if (size >= 0) len = size;
    else len = strlen (str);
    if (len > strlen(str)) return FALSE; // str is too short!
    int start = NextByte; // first character to be packed
    NextByte += len + 1;  // len for str, 1 for delimeter
    if (NextByte > MaxBytes) return FALSE;
    memcpy (&Buffer[start], str, len);
    Buffer [start+len] = Delim; // add delimeter
    BufferSize = NextByte;
    return TRUE;
}
```

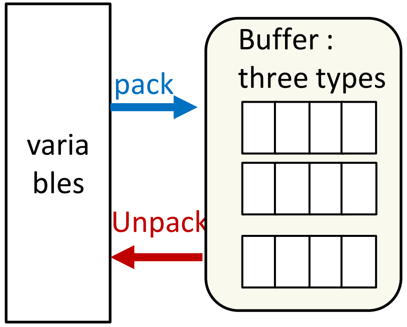
Pack() (2/2)





```
int DelimTextBuffer :: Pack (const char * str, int size)
{
    …
    int start = NextByte; // first character to be packed
    NextByte += len + 1;  // len for str, 1 for delimeter
    if (NextByte > MaxBytes) return FALSE;
    memcpy (&Buffer[start], str, len);
    Buffer [start+len] = Delim; // add delimeter
    BufferSize = NextByte;
    return TRUE;
}
```

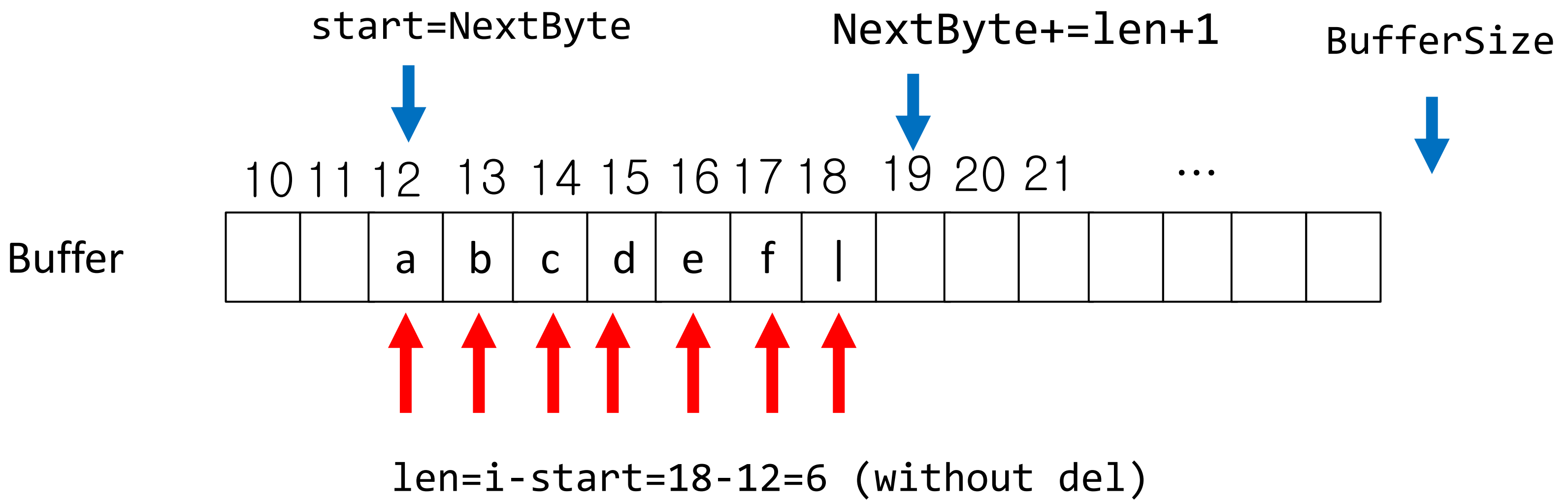
Unpack() (1/2)

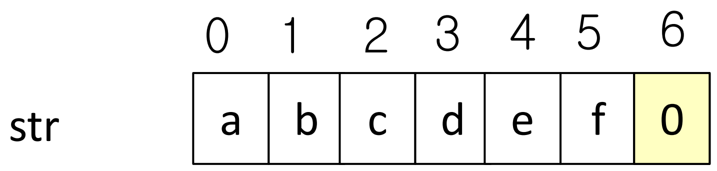


- Does not need a size
  - The field that is being unpacked consists of all of the characters up to the next instance of the delimeter

```
int DelimTextBuffer::Unpack(char *str)
// extract the value of the next field of the buffer
{
    int len = -1;                  // length of packed string
    int start = NextByte;          // first character to be unpacked
    for(int i = start; i < BufferSize; i++)
        if(Buffer[i] == Delim)     // next occurent of the delimeter
            {len = i-start; break;}    // compute the length by (i-start)
    if(len == -1) return FALSE; // delimiter not found
    NextByte += len + 1;
    if(NextByte > BufferSize) return FALSE;
    strncpy (str, &Buffer[start], len);
    str[len] = 0;   // zero termination for string
    return TRUE;
}
```

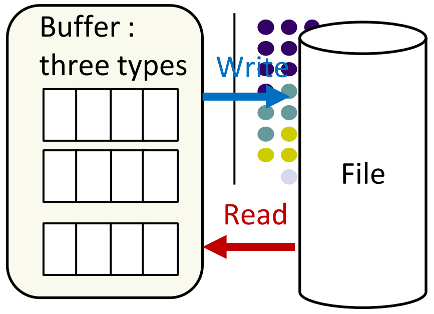
Unpack() (2/2)





```
int DelimTextBuffer::Unpack(char *str)
// extract the value of the next field of the buffer
{
    int len = -1;           // length of packed string
    int start = NextByte;   // first character
    for(int i = start; i < BufferSize; i++)
        if(Buffer[i] == Delim)   // next delimeter
            {len = i-start; break;} // compute the length
    if(len == -1) return FALSE; // delimiter not found
    NextByte += len + 1;
    if(NextByte > BufferSize) return FALSE;
    strncpy (str, &Buffer[start], len);
    str[len] = 0;   // zero termination for string
    return TRUE;
}
```

Read()

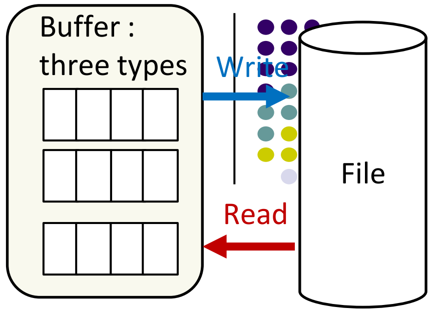


- Read() uses the variable-length strategy
  - clear the current buffer contents
  - extract the record size
  - read the proper number of bytes into the buffer
  - set the buffer size

```
void DelimTextBuffer::Clear()
{   // clear fields from buffer
    NextByte=0; BufferSize=0;
}

int DelimTextBuffer::Read(istream & stream)
{
    Clear();        // clear the current buffer
    // extract the recored size
    stream.read((char *)&BufferSize, sizeof(BufferSize));
    if (Stream.fail()) return FALSE;
    if (BufferSize > MaxBytes) return FALSE; // buffer overflow
    // read the proper number of bytes into the Buffer
    stream.read(Buffer, BufferSize);
    return stream.good();
}
```

Write()



- write()
  - write the size of a buffer first
  - write the contents of a buffer

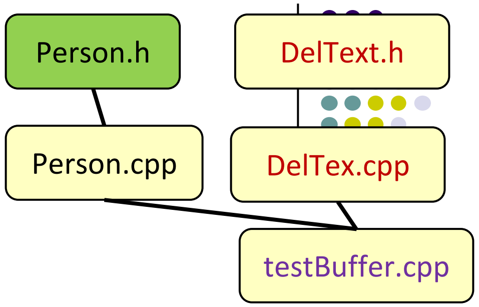
```
int DelimTextBuffer::Write(ostream & stream) const
{
    stream.write((char*)&BufferSize, sizeof(BufferSize));
    stream.write(Buffer, BufferSize);
    return stream.good();
}
```

Extending Class Person with Buffer Operations

- Buffer classes
  - pack and unpack any number and type of values
  - do not record how these values are combined to make objects

- To pack and unpack a buffer for a Person object
  - Specify the order in which the members of Person are packed and unpacked
  - for each record of Class Person, Pack() and Unpack() are called

Class Person

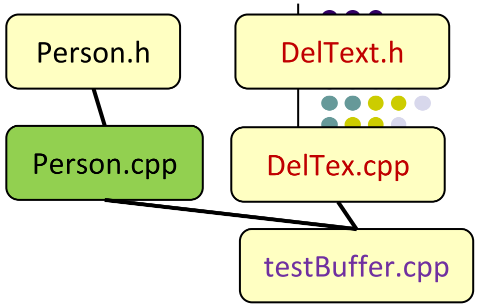


- Person.h

```
class Person
{
  public:
    // fields
    char LastName [11];   char FirstName [11];    char Address [16];
    char City [16];    char State [3];    char ZipCode [10];

    //operations
    Person ();
    void Clear ();
    static int InitBuffer (FixedTextBuffer &);
    int Unpack (FixedTextBuffer &);
    int Pack (FixedTextBuffer &) const;
    static int InitBuffer (LengthTextBuffer &);
    int Unpack (LengthTextBuffer &);
    int Pack (LengthTextBuffer &) const;
    static int InitBuffer (DelimTextBuffer &);
    int Unpack (DelimTextBuffer &);
    int Pack (DelimTextBuffer &) const;
    void Print (ostream &);
};
```

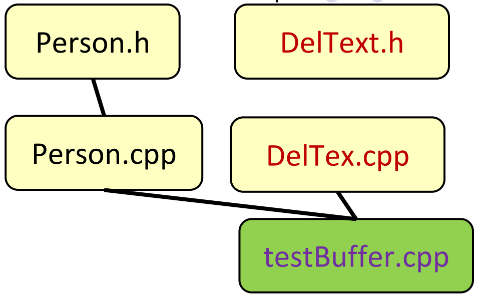
pack()/unpack() for Person



```
int Person::Pack (DelimTextBuffer & Buffer) const
{// pack the fields into a FixedTextBuffer,
 // return TRUE if all succeed, FALSE o/w
    int result;
    Buffer.Clear();
    result = Buffer.Pack(LastName);
    result = result && Buffer.Pack(FirstName);
    result = result && Buffer.Pack(Address);
    result = result && Buffer.Pack(City);
    result = result && Buffer.Pack(State);
    result = result && Buffer.Pack(ZipCode);
    return result;
}
```

```
int Person::Unpack (DelimTextBuffer & Buffer){
    int result;
    result = Buffer.Unpack (LastName);
    result = result && Buffer.Unpack(FirstName);
    result = result && Buffer.Unpack(Address);
    result = result && Buffer.Unpack(City);
    result = result && Buffer.Unpack(State);
    result = result && Buffer.Unpack(ZipCode);
    return result;
}
```

Test program for DelimTextBuffer (1/2)



```
int main(int argc, char ** argv)
{
    testDelText ();
}
```

```
void testDelText ()
{
    cout << "\nTesting DelimTextBuffer"<<endl;
    Person person;
    DelimTextBuffer Buff;
    Person::InitBuffer(Buff);
    strcpy(person.LastName, "Darling");
    strcpy(person.FirstName, "Pandora");
    strcpy(person.Address, "4112 Center St.");
    strcpy(person.City, "Tallahassee");
    strcpy(person.State, "FL");
    strcpy(person.ZipCode, "32306");
    person.Print(cout);

    Buff.Print(cout);
    cout <<"pack person "<< person.Pack(Buff)<<endl;
    Buff.Print (cout);

    ofstream TestOut("deltext.dat",ios::out);
    Buff.Write(TestOut);
```

Person object created

Buff before pack

Person is packed into Buff (1)

Buff after pack

Buff is written into “deltext.dat” (1)

Test program for DelimTextBuffer (2/2)

```
    strcpy(person.FirstName, "Dave");
    person.Print(cout);
    person.Pack(Buff);

    Buff.Write(TestOut);
    TestOut.close();

    ifstream TestIn("deltext.dat");
    DelimTextBuffer InBuff;
    Person :: InitBuffer(InBuff);
    cout <<"read "<<Buff.Read(TestIn)<<endl;
    cout <<"unpack "<<person.Unpack(Buff)<<endl;
    person.Print (cout);
    cout <<"read "<<Buff.Read(TestIn)<<endl;
    cout <<"unpack "<<person.Unpack(Buff)<<endl;
    person.Print (cout);
    cout <<"read "<<Buff.Read(TestIn)<<endl;
    cout <<"unpack "<<person.Unpack(Buff)<<endl;
    person.Print (cout);
}
```

Person has been modified

Person is packed into Buff (2)

Buff is written into “deltext.dat” (2)

A Person object load into Buff

Person is unpacked from Buff

Makefile

```
CFLAGS= -Wall
OBJS = Person.o Deltext.o testBuffer.o
all: testBuffer
%.o: %.cpp
	g++ -c -o $@ $(CFLAGS) $<
testBuffer: $(OBJS)
	g++ -o testBuffer $(OBJS)
clean:
	-rm -rf testBuffer $(OBJS)
```

Execute program

```
$ ./testBuffer
Testing DelimTextBuffer
Person:         Last Name 'Darling'
               First Name 'Pandora'
                 Address '4112 Center St.'
                   City 'Tallahassee'
                  State 'FL'
                Zip Code '32306'
Buffer has max characters 1000 and Buffer Size 0
pack person 1
Buffer has max characters 1000 and Buffer Size 53
Person:         Last Name 'Darling'
               First Name 'Dave'
                 Address '4112 Center St.'
                   City 'Tallahassee'
                  State 'FL'
                Zip Code '32306'
```

```
read 1
unpack 1
Person:         Last Name 'Darling'
               First Name 'Pandora'
                 Address '4112 Center St.'
                   City 'Tallahassee'
                  State 'FL'
                Zip Code '32306'
read 1
unpack 1
Person:         Last Name 'Darling'
               First Name 'Dave'
                 Address '4112 Center St.'
                   City 'Tallahassee'
                  State 'FL'
                Zip Code '32306'
read 0
unpack 0
Person:         Last Name 'Darling'
               First Name 'Dave'
                 Address '4112 Center St.'
                   City 'Tallahassee'
                  State 'FL'
                Zip Code '32306'
```

Bug?

Outline

- 4.1 Field and Record Organization
- 4.2 Using Classes to Manipulate Buffers
  - Buffer class for delimited fields
  - Buffer class for length-based fields
  - Buffer class for fixed-length fields

- 4.3 Using Inheritance for Record Buffer Classes
- 4.4 Managing Fixed-Length, Fixed-Field Buffers
- 4.5 An Object-Oriented Class for Record Files

Buffer classes for length-based and fixed-length fields

- representing records of length-based fields and records of fixed-length fields
  - need a change in the Pack and Unpack methods of the delimited field class
  - the class definitions are almost exactly the same

class LengthTextBuffer

```
// a buffer which holds length-based text fields.
class LengthTextBuffer
{
public:
    LengthTextBuffer (int maxBytes = 1000);
    // construct with a maximum of maxBytes
    void Clear (); // clear fields from buffer
    int Read (istream &);
    int Write (ostream &) const;
    int Pack (const char *, short size = -1);
    // set the value of the next field of the buffer;
    int Unpack (char *);
    // extract the value of the next field of the buffer
    void Print (ostream &) const;
    int Init (int maxBytes = 1000);
private:
    char * Buffer; // character array to hold field values
    int BufferSize; // size of packed fields
    int MaxBytes; // maximum number of characters in the buffer
    int NextByte; // packing/unpacking position in buffer
};
```

No delim parameter

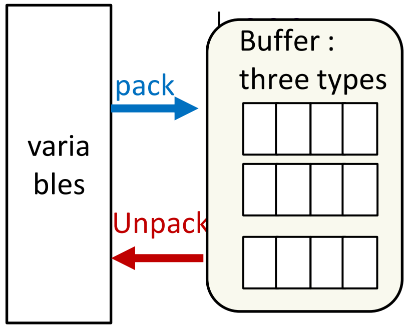
Member variable for delim is removed

LengthTextBuffer::Clear() and Init()

```
void LengthTextBuffer :: Clear ()
// clear fields from buffer
{
    NextByte = 0;
    BufferSize = 0;
}

int LengthTextBuffer :: Init (int maxBytes)
 // construct with a maximum of maxFields
{
    if (maxBytes < 0) maxBytes = 0;
    MaxBytes = maxBytes;
    Buffer = new char[MaxBytes];
    Clear ();
    return 1;
}
```

LengthTextBuffer::Pack()

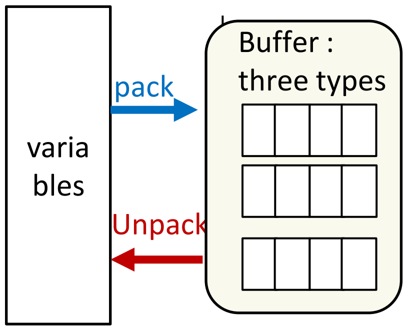


```
int LengthTextBuffer::Pack (const char * str, short size)
// set the value of the next field of the buffer;
// if size = -1 (default) use strlen(str) as length of field
{
    short len; // length of string to be packed
    if (size >= 0) len = size;
    else len = strlen (str);
    if (len > strlen(str)) // str is too short!
        return FALSE;
    int start = NextByte; // first character to be packed
    NextByte += (len + sizeof(len));
    if (NextByte > MaxBytes) return FALSE;
    memcpy (&Buffer[start], &len, sizeof(len));
    strncpy (&Buffer[start+sizeof(len)], str, len);
    BufferSize = NextByte;
    return TRUE;
}
```

Length first

then, string

LengthTextBuffer::Unpack()



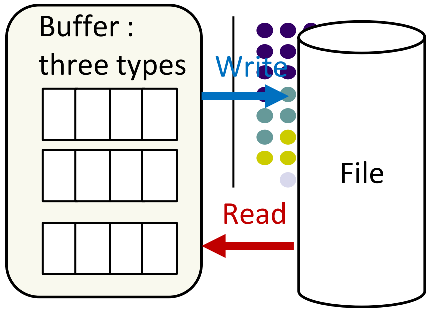
- 1. read a length from Buffer
- 2. copy a string with a length from buffer to str

```
int LengthTextBuffer::Unpack(char * str)
// extract the value of the next field of the buffer
{
    short len; // length of packed string
    if (NextByte >= BufferSize) return FALSE; // no more fields
    int start = NextByte; // first character to be unpacked
    memcpy (&len, &Buffer[start], sizeof(short));
    NextByte += len + sizeof(short);
    if (NextByte > BufferSize) return FALSE;
    strncpy (str, &Buffer[start+sizeof(short)], len);
    str [len] = 0; // zero termination for string
    return TRUE;
}
```

Length first

then, string

LengthTextBuffer::read/write



- Write()
  - Same with the Class DelimTextBuffer::write()

```
int LengthTextBuffer::Write(ostream & stream) const
{
    stream.write ((char*)&BufferSize, sizeof(BufferSize));
    stream.write (Buffer, BufferSize);
    return stream.good ();
}
```

- Read(): Buffer is cleared before read()

```
int LengthTextBuffer::Read(istream & stream)
{
    Clear();
    stream.read((char *)&BufferSize, sizeof(BufferSize));
    if (stream.fail()) return FALSE;
    if (BufferSize > MaxBytes) return FALSE; // buffer overflow
    stream.read(Buffer, BufferSize);
    return stream.good();
}
```

Outline

- 4.1 Field and Record Organization
- 4.2 Using Classes to Manipulate Buffers
  - Buffer class for delimited fields
  - Buffer class for length-based fields
  - Buffer class for fixed-length fields
- 4.3 Using Inheritance for Record Buffer Classes
- 4.4 Managing Fixed-Length, Fixed-Field Buffers
- 4.5 An Object-Oriented Class for Record Files

class FixedTextBuffer (1/3)

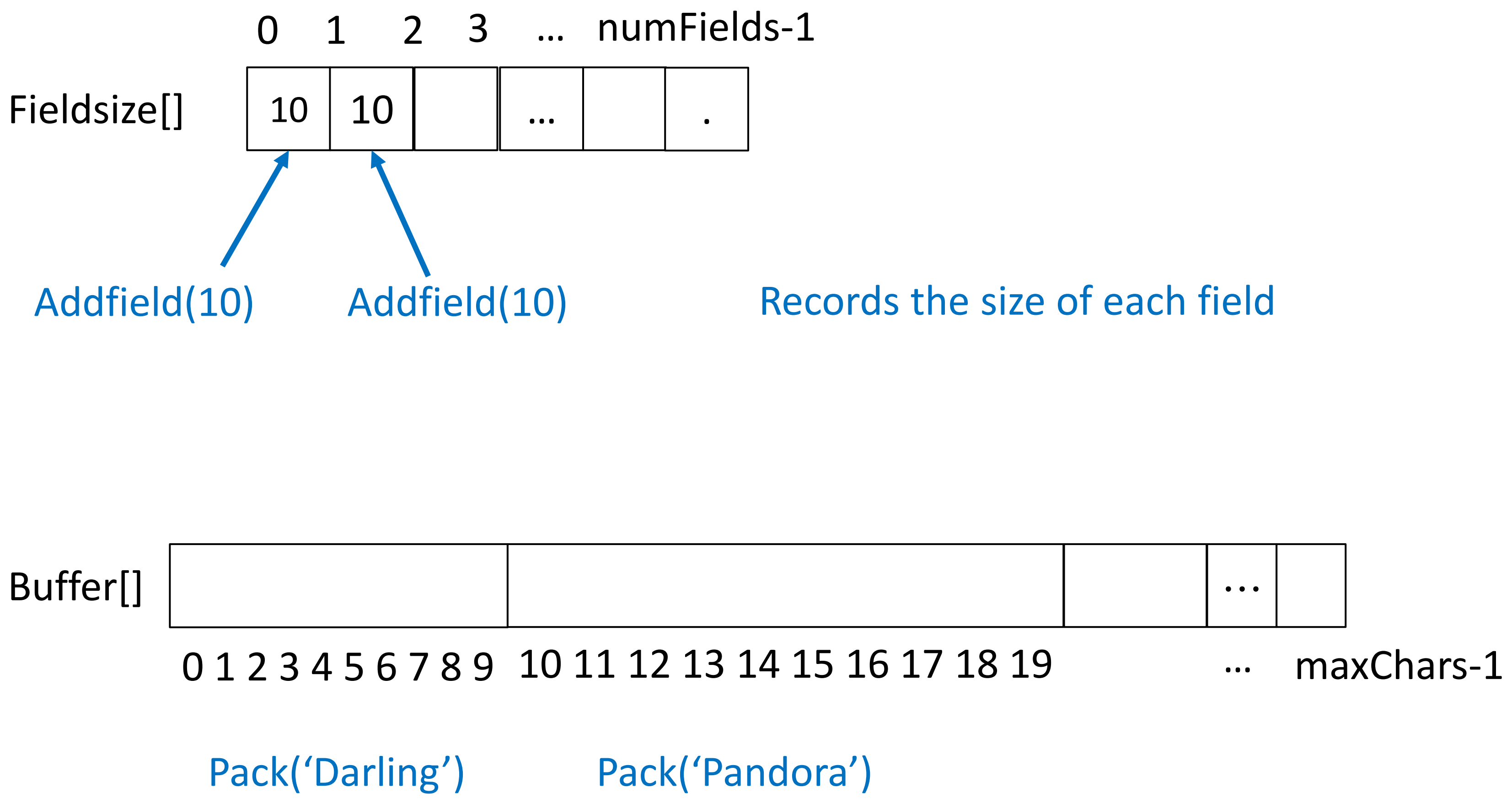
```
class FixedTextBuffer
// a buffer which holds a specific number of fixed sized text fields.
{  public:
    FixedTextBuffer (int maxFields, int maxChars = 1000); // construct with a maximum of maxFields
    FixedTextBuffer (int numFields, int * FieldSize);
        // construct with fields of specific size
    int NumberOfFields () const; // return number of fields
    void Clear (); // clear field values from buffer
    int AddField (int fieldSize);
    int Read (istream &);
    int Write (ostream &);
    int Pack (const char *); // set the value of the next field of the buffer;
    int Unpack (char *); // extract the value of the next field of the buffer
    void Print (ostream &);
    int Init (int numFields, int maxChars = 1000);
    int Init (int numFields, int * fieldSize);
 private:
    char * Buffer; // character array to hold field values
    int BufferSize; // sum of the sizes of declared fields
    int * FieldSize; // array to hold field sizes
    int MaxFields; // maximum number of fields
    int MaxChars; // maximum number of characters in the buffer
    int NumFields; // actual number of defined fields
    int NextField; // index of next field to be packed/unpacked
    int NumFieldValues; // number of fields which are packed
    int Packing; // TRUE if in packing phase, FALSE o/w
    int NextCharacter; // packing/unpacking position in buffer
};
```

class FixedTextBuffer (3/3)

- use a fixed collection of fixed-length fields
- pack() needs no size parameter
- use fixed-length records
- read() and write() do not use a length indicator for buffer size

class FixedTextBuffer (3/3)

- Conceptual view



Person:InitBuffer()

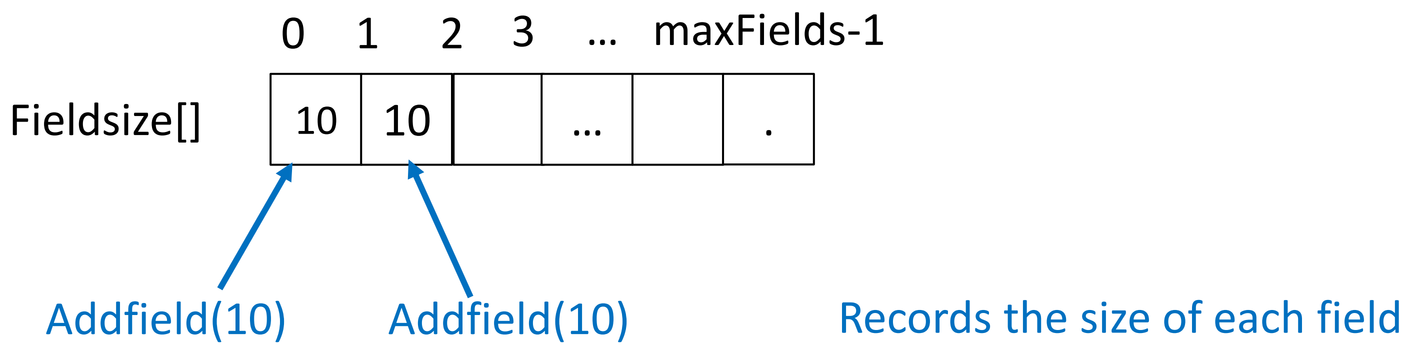
```
int Person::InitBuffer(FixedTextBuffer & Buffer)
// initialize a FixedTextBuffer to be used for Persons
{
    int result;
    result = Buffer.AddField (10); // LastName [11];
    result = result && Buffer.AddField (10); // FirstName [11];
    result = result && Buffer.AddField (15); // Address [16];
    result = result && Buffer.AddField (15); // City [16];
    result = result && Buffer.AddField (2);  // State [3];
    result = result && Buffer.AddField (9); // ZipCode [10];
    return result;
}
```

FixedTextBuffer :: Init()

```
int FixedTextBuffer::Init (int maxFields, int maxChars)
 // construct with a maximum of maxFields
{
    if (maxFields < 0) maxFields = 0;
    if (maxChars < 0) maxChars = 0;
    MaxFields = maxFields;
    MaxChars = maxChars;
    FieldSize = new int[MaxFields];
    Buffer = new char[MaxChars];
    BufferSize = 0;
    NumFields = 0;
    NextField = 0;
    Packing = TRUE;
    return 1;
}
```

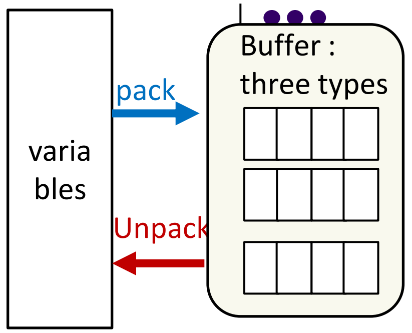
FixedTextBuffer :: Addfield()

- supports the specification of the fields and their sizes

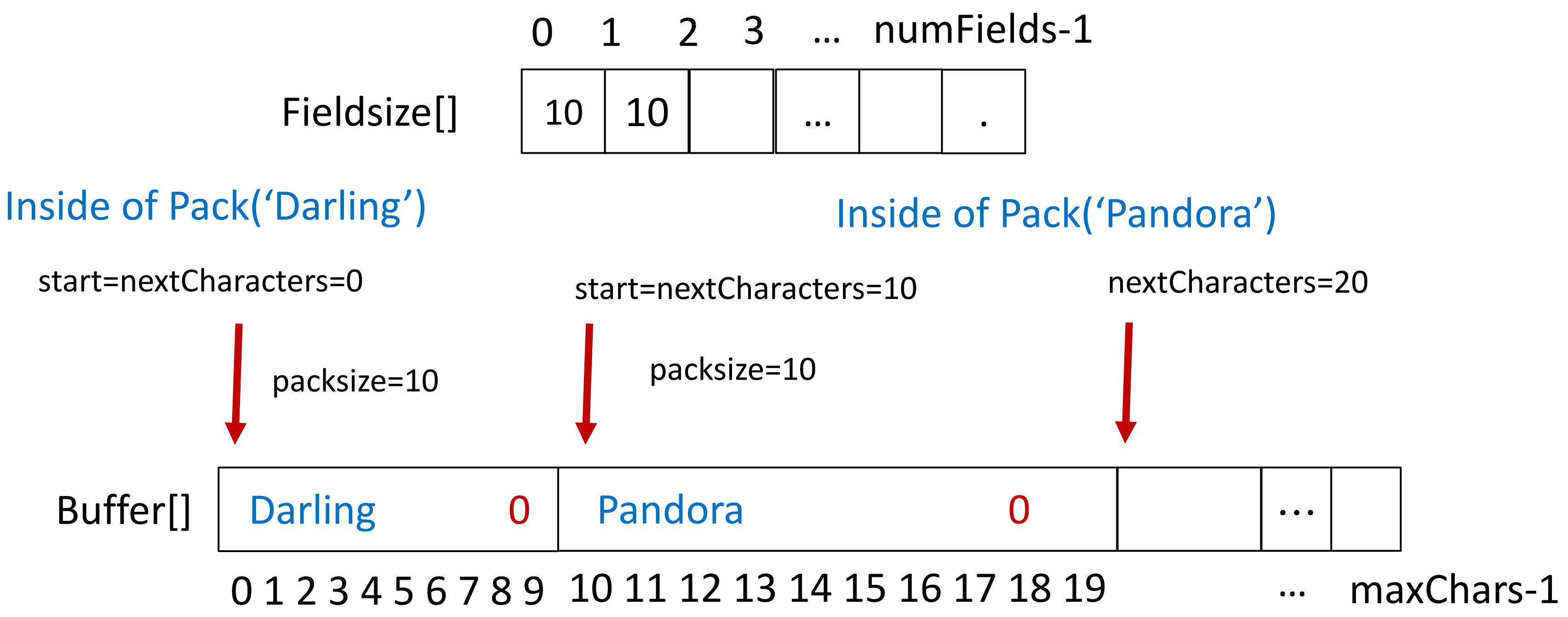


```
int FixedTextBuffer::AddField (int fieldSize)
{
    if (NumFields == MaxFields) return FALSE;
    if (BufferSize + fieldSize > MaxChars) return FALSE;
    FieldSize[NumFields] = fieldSize;
    NumFields ++;
    BufferSize += fieldSize;
    return TRUE;
}
```

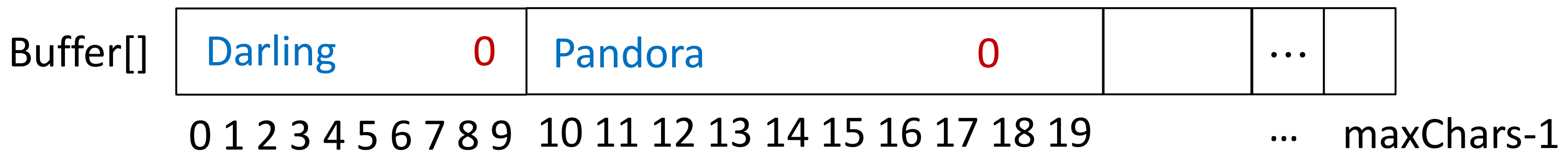
FixedTextBuffer::Pack() (1/2)



- Fieldsize[] let us know the size of each field
- NextCharacters keeps the position of the next field



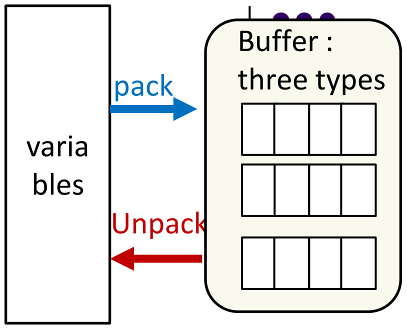
FixedTextBuffer::Pack() (2/2)



```
int FixedTextBuffer::Pack(const char * str)
// set the value of the next field of the buffer;
{
    if (NextField == NumFields || !Packing) // buffer is full or not packing mode
        return FALSE;
    int len = strlen(str);
    int start = NextCharacter; // first byte to be packed
    int packSize = FieldSize[NextField]; // number bytes to be packed
    strncpy(&Buffer[start], str, packSize);
    NextCharacter += packSize;
    NextField ++;
    // if len < packSize, pad with blanks
    for (int i = start + packSize; i < NextCharacter; i ++)
        Buffer[start] = ' ';
    Buffer[NextCharacter] = 0; // make buffer look like a string
    if (NextField == NumFields) // buffer is full
    {
        Packing = FALSE;
        NextField = NextCharacter = 0;
    }
    return TRUE;
}
```

Buggy code!

FixedTextBuffer::Unpack()



```
int FixedTextBuffer::Unpack (char * str)
// extract the value of the next field of the buffer
{
    // buffer is full or not unpacking mode
    if (NextField == NumFields || Packing)
        return FALSE;
    int start = NextCharacter; // first byte to be unpacked
    int packSize = FieldSize[NextField]; // number bytes to be unpacked
    strncpy (str, &Buffer[start], packSize);
    str [packSize] = 0; // terminate string with zero
    NextCharacter += packSize;
    NextField ++;
    if (NextField == NumFields) Clear (); // all fields unpacked
    return TRUE;
}
```

Q&A